AD-A053 315 MOORE SCHOOL OF ELECTRICAL ENGINEERING PHILADELPHIA P--ETC F/G 14/2

NOPAL PROCESSOR: INTRA-TEST SEQUENCING.(U)

JAN 78 R W BERMAN DAAA25-75-C-0650

UNCLASSIFIED ECOM-75-0650-F NL

1 OF 2

AD A053315

(12)

Research and Development Technical Report
ECOM 75-0650-F

NOPAL PROCESSOR: INTRA-TEST SEQUENCING

Ronald Wayne Berman
Moore School of Electrical Engineering
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, Pennsylvania 19104

January 1978
Final Report for Period 30 June 1975 - 31 August 1977

PREPARED FOR:

ECOM

US ARMY ELECTRONICS COMMAND FORT MONMOUTH, NEW JERSEY 07703

NOTICES

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

8 19 REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ECOM 75-Ø65Ø-F	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 NOPAL PROCESSOR: INTRA-TEST SEQUENCING.		5. TYPE OF REPORT & PERIOD COVERED 9 FINAL Report. 30 Jun 75 - 31 Aug 77 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) 10 Ronald Wayne/Berman		8. CONTRACT OR GRANT NUMBER(s) 15 DAAA25-75-C-Ø65Ø DAAA-25-75-C0650
9. PERFORMING ORGANIZATION NAME AND ADDRESS Moore School of Electrical Engineering Department of Computer and Information Science University of Pennsylvania, Philadelphia, PA 19104		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS [redacted] 16 1G6 63622 AJ29 17 ØØ 001
11. CONTROLLING OFFICE NAME AND ADDRESS US Army Electronics Command ATTN: DRSEL-TL-MS Fort Monmouth, NJ 07703		12. REPORT DATE 11 Jan[redacted]78 13. NUMBER OF PAGES 100 12 1Ø6p.
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES The work covered by this report was sponsored by Frankford Arsenal, US Army, Philadelphia. However, since the Frankford Arsenal mission relative to this work was transferred to the US Army Electronics Command (ECOM), Ft. Monmouth, NJ, this report is being issued as an ECOM report.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Automatic Testing
Automatic Test Systems
Automatic Test Equipment
Automatic Generation of Test Programs

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes the algorithms, methods, and programs that represent the intra test sequencing phase of the NOPAL processor. The overall objective of the NOPAL system is to generate a program for computer controlled automatic test equipment for testing an electronic unit under test. The input to the system is a nonprocedural description of the desired test expressed in the NOPAL language. The present report describes how such a specification can be analyzed and how a flowchart can be generated for performing the test. A subsequent phase of NOPAL generates a program based on the flowchart. — (continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

410 039

[illegible]

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

20. ABSTRACT (continued)

The report can be read independently of the NOPAL processor. Readers interested only in the automatic sequencing of operations in a program can omit Chapters 1 and 2, which describe the NOPAL language and processor. Readers interested in greater detail on the NOPAL processor can obtain further information in the references listed in the bibliography.

CONTENTS

Page

FIGURES:

1 INTRODUCTION

Computer controlled Automatic Test Equipment (ATE) increases the efficiency while reducing the cost of testing complex equipment. The function of the computer program which operates the ATE is to validate the Unit Under Test(UUT) design, test point availability and packaging. The complex computer program logic needed to specify test sequences require the programmer to have expert knowledge of the Unit Under Test (UUT) and of computer technology (programming). The NOPAL language was developed to simplify the task of specifying test sequences controlling the ATE.

NOPAL is an acronym for "Nonprocedural Operational Performance Analysis Language". Independent NOPAL test modules each containing the logic and diagnostic instructions for a particular test situation may be written in nonprocedural or random order. A significant advancement in NOPAL over other test specifications languages is the ability to automatically sequence the test modules, and the instructions within the test modules. This attribute provides the flexibility to modify test instructions without manually reordering other statements or test modules.

The report presents the algorithms, methods, and computer programs used to automatically order the statements within a test module.

1.1 BACKGROUND:

Maintenance and support expense for complex equipment are approaching procurement costs in many organizations. For example, it is reported that of the fifteen and a third billion dollars spent on electronics by the United States Department of Defense, more than one third, or five and a quarter billion dollars, was spent on maintenance and support [Eleccion, 1974]. It is estimated that 80% of this cost is attributed to labor, with 85% of the time spent on diagnosis and fault isolation. Fiscal savings can be achieved by utilizing engineering technicians with Automatic Test Systems (ATS). This may be contrasted to the present manual procedures employed by electical engineers to determine optimal test sequences.

The probable savings can be calculated by comparing the beginning salary of engineers and engineering technicians as reported by the United States Department of Labor Statistics [U.S. BLS, 1975)]. In 1974, beginning engineers received $13,171 per year as contrasted to the engineering technician's $7,975 per year, a difference of forty percent. Based on the maintenance and support figures given above, the estimated savings might be one and a half billion dollars, assuming there is a one-to-one substitution ratio between the engineering technicians and the electrical engineers. The savings would enable electrical engineers to utilize their talents more productively.

To achieve fiscal economy and enhanced reliability as a result of effective testing, the United States Army developed a special purpose language, OPAL (Operational Performance Analysis Language) [Frankford Arsenal, 1976] to mechanize the programming of ATS. The design is general enough to test a broad range of equipment including electronic, mechanical, hydraulic, and optical. Its operation is similar to the well known computer language, BASIC, in that each line begins with a keyword such as DECLARE, GOTO, CALL, etc. OPAL's most significant improvement over earlier test programming languages such as ATLAS [ARINC, 1972] is its facility for modular development of test modules, allowing independent test development by different programming teams.

Use of OPAL as a programming language requires considerable programming labor. For example, the exact execution sequence of instructions within each test module and the overall ordering of test modules must be specified. This is done with the conventional GOTO and CALL statements. A high likelihood of error is present when manually implementing the fault determination strategy. This is a result of the extensive knowledge needed in computer programming, applied mathematics, and the component being tested. Lengthy sequencing instructions combined with required storage assignments, produce multiple coding steps for all test specifications. In order to simplify the process of programming the ATS, a non-procedural test specification language was developed at the University of Pennsylvania in the Moore School of Electrical Engineering, Department of Computer and Information Sciences [Che, 1976].

NOPAL - an acronym for Nonprocedural Operational Performance Analysis Language permits the user to specify test specifications in modular fashion, independent of one another. The actual sequence of execution is determined automatically by the NOPAL processor. Since the sequencing is performed automatically, all GOTO's and subroutine CALL's have been eliminated. The name NOPAL was selected because it is the name of a cactaceous plant which illustrates an incremental growth of stems analogous to the modular development of tests in NOPAL.

Unlike OPAL, NOPAL is not a programming language. Rather, it is a language for describing individual tests and diagnoses which are used as input to the NOPAL processor. The OPAL processor then produces a computer program for conducting these tests in conjunction with Automatic Test Equipment (ATE). Figure 1.1 illustrates this relationship.

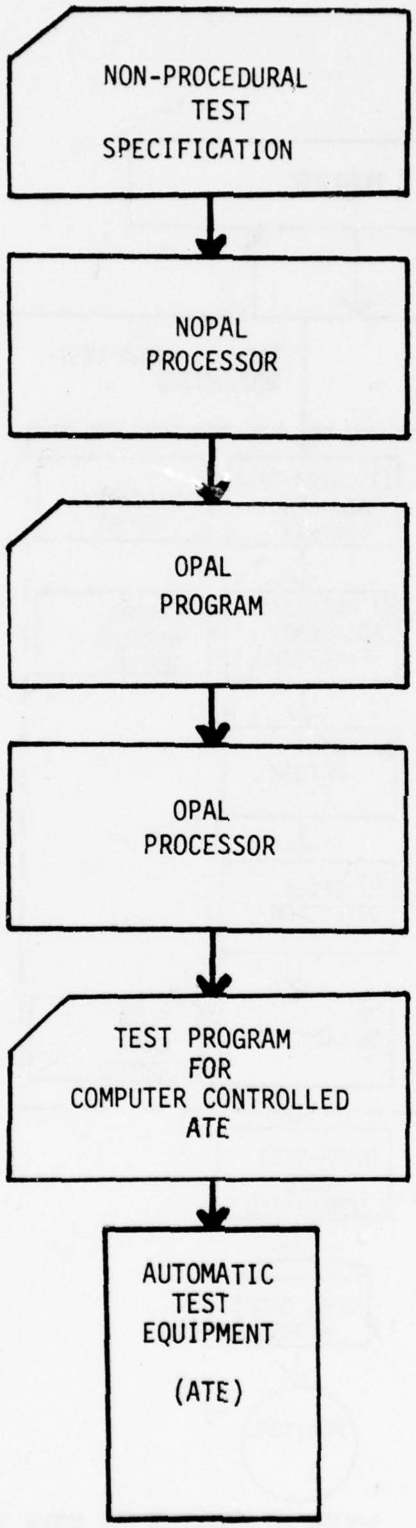
NOPAL's non-proceduralness also allows for incrementality in the sense that additions or modifications to test specifications may be incorporated easily. For example, when tests are added because of design modifications, the user need not construct alternative sequencing as in OPAL. The automatic ability to sequence enables NOPAL to achieve a degree of sophistication that will reduce the cost of updating and creating the program for the ATS.

Another improvement incorporated in NOPAL is its ability to automatically allocate storage assignments for single and subscript variables. These assignments indicate whether a variable is locally defined to a particular test module or globally defined to all test modules. The attribute, whether local or global, is stated in several user reports.

1.2 THE NOPAL PROCESSOR

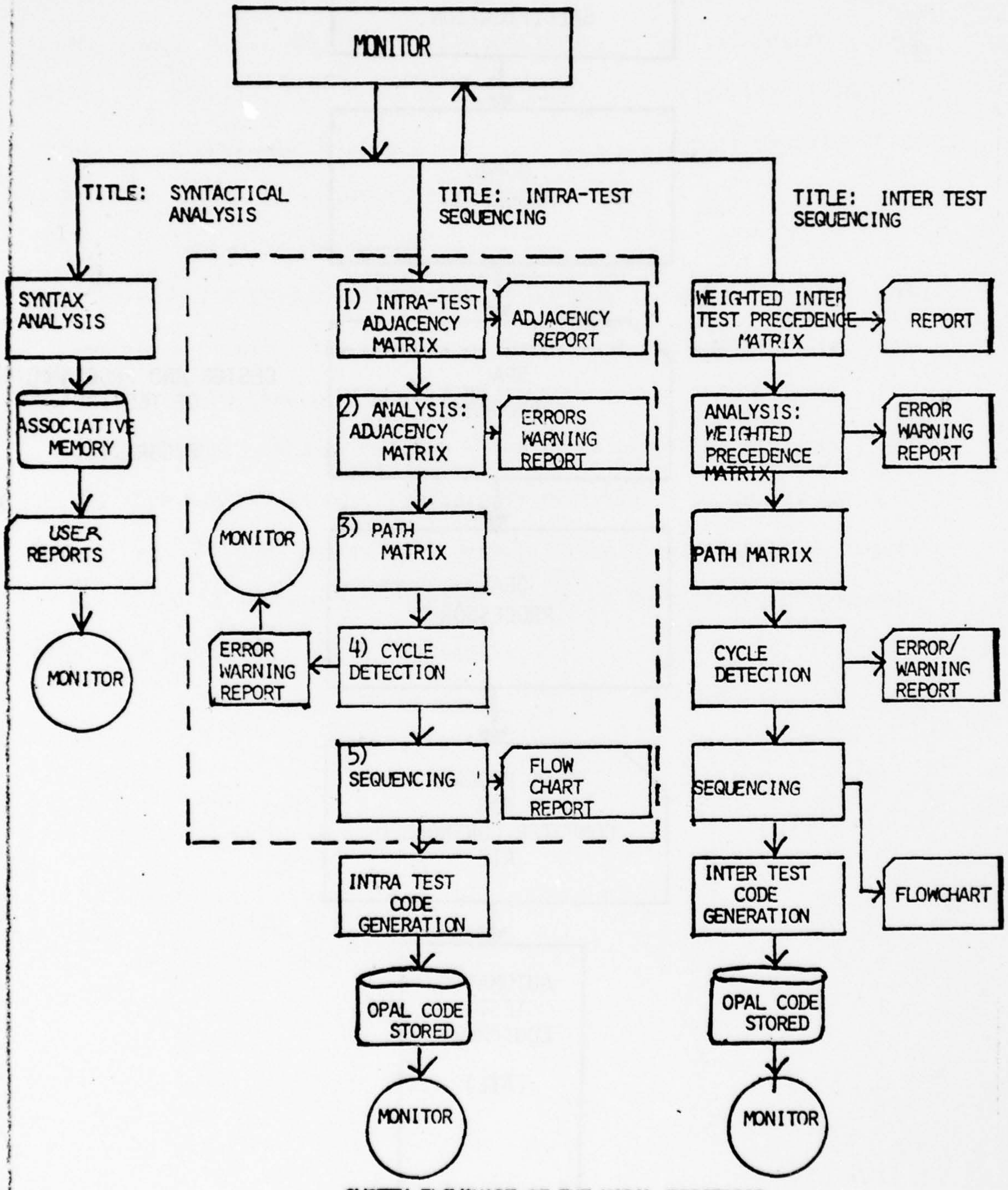
The NOPAL processor is graphically represented in Figure 1.2. The monitor, located at the top of this diagram, is the control mechanism of the system involving the procedures below it. These procedures, titled "syntax analysis," "intra-test analysis," and "inter-test analysis," form the three basic divisions of the processor.

Syntax analysis, drawn on the far left portion of Figure 1.2, refers to the process of parsing NOPAL source strings into syntactic


NON-PROCEDURAL TEST SPECIFICATION
NOPAL PROCESSOR
OPAL PROGRAM
OPAL PROCESSOR
TEST PROGRAM FOR COMPUTER CONTROLLED ATE
AUTOMATIC TEST EQUIPMENT (ATE)

DESIGN AND PROGRAMMING OF TESTING

FIGURE 1.1



SYSTEM FLOWCHART OF THE NOPAL PROCESSOR

FIGURE I.2

classes, and storing them in an associative memory. This special purpose memory network stores the strings so that they are "associated" by their content, and can be retrieved easily during later stages of processing. User reports including a source listing, a cross reference report, and an error report are produced during this phase.

Intra-test analysis is represented in the center of Figure 1.2 partitioned by the broken rectangle. Its purpose is to evaluate expressions within each test module to determine automatically their sequence of execution. The process begins by establishing an adjacency matrix, for every NOPAL test module. Each position of the adjacency matrix represents a variable name, a diagnosis, or a conjunction or assertion. Relations are represented on the matrix such that source and target variables are linked to conjunction and assertions, or operator response and other parameter variables are linked to diagnoses. A path matrix is later introduced to detect circular definitions by indicating the existence of all paths within the NOPAL test module. If the NOPAL test module is error free, sequencing is performed using precedence-and-ranking-algorithms based on graph theory. Reports listing the sequenced NOPAL statements and the detected errors are generated.

Inter-test analysis, drawn on the right portion of Figure 1.2, refers to the process of automatically determining the order of execution for all test modules. Six sequencing strategies are combined with precedence relationship rules to form a directed graph, represented as a precedence matrix. Each position of the precedence matrix represents a variable name, diagnosis, or test module. The row and column combinations denote a precedence relationship. Based on the matrix, it can be determined whether the test specifications are complete, consistent and/or unambiguous. Also, possible cycles are detected and eliminated appropriately. Finally, test modules are assigned execution levels and ordered in proper execution sequence. A detailed report explaining this process is currently in progress.

1.3 Organization of the Report

Section 2 introduces the types of NOPAL statements. Following this, the sections parallel the process of intra-test sequencing illustrated in the center portion of Figure 1.2.

In Section 3, the algorithms which transform NOPAL statements into matrix form are described. Preliminary analysis of ambiguity is in Section 4. The path matrix that detects circular or transitive relations is presented in Section 5. Finally, Section 6 details the precedence algorithm that orders NOPAL statements.

2. OVERVIEW OF NOPAL STATEMENT TYPES

NOPAL statements concisely express test situations whose sequencing is discussed in this report. This section presents an overview of the language to familiarize the reader with its construction. To actually write a test module, the NOPAL Language Manual should be used.

The overall structure of the NOPAL language is summarized in Figure 2.1, using an extended BNF notation. As shown at the top line, a specification in NOPAL has three parts: Test, Unit Under Test (UUT), and ATE. The latter two are intended to provide UUT and ATE independence, in the sense that changes are reflected only in the corresponding parts of the specification. Consistent with the theme of this report, the discussion centers around the format of the NOPAL test specifications written in independent test modules.

A test module describes a single test to diagnose a variety of failure modes. Because the modules are independent of each other, they may be modified, deleted, or inserted without affecting other test situations. Formally, a test module is a composite of four unique classes: stimuli, measurement, logic, and diagnosis.

The Stimuli class indicates the actions applied to the UUT at test time, while the Measurements class expresses what actions are needed to ascertain the success or failure of the test. Both stimuli and measurement employ conjunction and assertion statements. A conjunction indicates a function applied or measured to a specific connector point of the UUT. An Assertion may be used for two purposes. The primary one is for stating the range of measurements which would determine the success or failure of a test. The other use is to

```
< NOPAL SPECIFICATION > ::= < TEST SPECIFICATION > < UUT SPECIFICATION >
                            < ATE SPECIFICATION >

< TEST SPECIFICATION > ::= < TEST MODULE > [ < TEST MODULE > ]*

< TEST MODULE > = [ < STIMULI > ] [ < MEASUREMENT > ]
                  [ < LOGIC > ]* [ < DIAGNOSIS > ]*

< STIMULI > ::= [ < CONJUNCTION > ] [ < ASSERTION > ]*

< MEASUREMENT > ::= [ < CONJUNCTION > ] [ < ASSERTION > ]*

< CONJUNCTION > ::= < TEST POINTS > < RELATION > < WAVEFORM >
                    [ < TEST POINTS > < RELATION > < WAVEFORM > ]*

< ASSERTION > ::= [IF CLAUSE THEN]
                  < ARITH_EXPR > < RELATION > < ARITH_EXPR >
                  [ELSE < ARITH_EXPR > < RELATION > < ARITH_EXPR >]

< LOGIC > ::= < OPERATOR > < DIAGNOSIS ID >

< DIAGNOSIS > ::= < DIAGNOSIS ID > < MESSAGE > [ < FAILURE IDS > ]
                  [ < OTHER DATA > ] [ < TIMING > ]
                  [ < OPERATOR RESPONSES > ]

Key:  [< - >]  OPTIONAL

      [< - >]* MAY REPEAT ZERO OR MORE TIMES
```

FIGURE 2.1 TOP LEVEL STRUCTURE OF THE NOPAL LANGUAGE

store data values and perform computation during execution of the testing process.

The sample test module (Figure 2.2) demonstrates the use of stimuli, measurement, logic, and diagnosis to express a test situation. The only stimuli is conjunction $S_W0001 which applies function "CONST_S" to connector pins "J24_B" and "GND". To measure the result of $S_W0001, measurement conjunction $M_W0001 applies function "SINE_D" to connector pins "J22" and "GND". Finally measurement assertion $M_W0002 evaluates the results to determine the success or failure of the test.

Conjunctions and assertions employ two classes of variables: Source and target. Source variables are generated elsewhere, in other assertions or conjunctions. TARGET variables are locally evaluated to be used elsewhere. Source variables may be considered dependent or exogenous; while target variables may be considered independent or endogenous. Within the sample test module (Figure 2.2), conjunction $M_W0001 defines "F1" and "V1" as target variables, and "VAR1" as a source variable. Assertion $M_W0002 defines "VAR1" and "F1" as source variables. Essentially a variable defined both as source and target by different conjunctions and assertions causes the conjunction or assertion which uses the variable as target to automatically precede the conjunction or assertion that employs it as source.

Each diagnosis identifies the failures and the message which communicates the results of testing to the operator. A diagnosis statement is composed of five parts: affected components, other parameters, message type, timing, and operator response. Affected components is a list of

```
TEST SYTEST0003;

   STIMULI 2(SYTEST0003);

      CONJUNCTION $S_W0001(2):
         ( < J24_B > GND = CONST_S(27.5 VOLT ));

   MEASUREMENT $M_SYTEST000(SYTEST0003);

      CONJUNCTION  $M W0001($M SYTEST000):
         ( J22, GND   = SINE_D(V1 VOLT, F1 HZ, VAR1 SEC ))
                    TARGET:  F1, V1
                    SOURCE:  VAR1;

      ASSERTION  $M_W0002($M_SYTEST000):

         IF VAR1=60 THEN
            F1 = 5*1E+06 +- 60
         ELSE
            F1 = 5*1E+06 +- 2.5
                   SOURCE:  VAR1, F1;

LOGIC $LOGIC0010(SYTEST003):  *4,   5, *6;

  DIAGNOSIS 4:
     OPERATOR MESSAGE:
        TYPE=#5,
        TIME= 0.00000E+00SEC,
     RESPONSE=(VAR1);

  DIAGNOSIS 5:
     OPERATOR MESSAGE:
        AFFECTED COMPONENTS=FREQ TOL(STD_5MHZ_FRE),
        OTHER PARAMETERS=('FREQ'),
        TYPE=#6;

  DIAGNOSIS 6:
     OPERATOR MESSAGE:
        OTHER PARAMETERS=(F1, 'HZ'),
        TYPE=D;
```

FIGURE 2.2 · SAMPLE TEST MODULE

failure functions indicating the modes of failure and corresponding components which the diagnosis asserts to have failed. Other parameters indicates the variables or character strings that are included in the diagnostic message. The diagnostic message is referred to by the label in the message type. Timing states when the operator message is to be sent in respect to the beginning of the application of the stimuli. If the message contains instructions to the operator to perform duties such as pressing keys, reading meters, or making measurements, an operator response may be necessary in order to conclude the tests. The possible responses consist of suspending or initiating a test or inputting to the terminal the values of the requested variables.

Operator response and other parameters qualify the diagnostic message. Figure 2.2 presents three diagnoses. Diagnosis 4 uses VAR1 to store the operator response for latter examination. Diagnosis 5 employs the character string 'FREQ' as an other parameter. Diagnosis 6 uses variables F1 and character string 'HZ' as an other parameter.

The next section discusses how the information necessary for sequencing is constructed from the test module and represented in a matrix equivalent form.

3. ADJACENCY MATRIX

This phase of the NOPAL processor deals with the transformation of NOPAL test specifications by the use of graphs and matrices. It describes an application of graph theory to the analysis of NOPAL specifications and to the generation of a sequenced test module. While graph theory has been used in various computer applications in recent years the analysis of information relationships and the automatic sequencing by means of graph are novel. This section presents the background and terminology involved, as well as describing the graphs, matrices, and other data structures that are built from a NOPAL specification test module for intra-test sequencing.

Section 3.1 provides an overview of the processes involved in this stage, and Section 3.2 discusses them in greater detail.

3.1 ELEMENTARY STRUCTURES OF THE ADJACENCY MATRIX

This section discusses a matrix representation of the test specifications introduced in Section 2. Of these specifications, the precedence indicator controlling sequencing is based on source and target variables. For example, a source variable to an assertion must be available before invoking the assertion. A target variable to an assertion is only available after the assertion is computed. This requirement detailing precedence information is implied in a "directed graph".

A directed graph is a network of interconnected nodes such that each node is a conjunction, assertion, diagnosis, or variable. Formally, a directed graph is a pair (N,A) such that N represents the set of nodes in the test module; and A represents the set of ordered pairs (Nj, Nk). Each Ai is illustrated with an arrow indicating a path from node Nj to node Nk. Each node may have multiple arrows emanating from it. Because the arrows have an associated direction the graph is sometimes called a digraph.

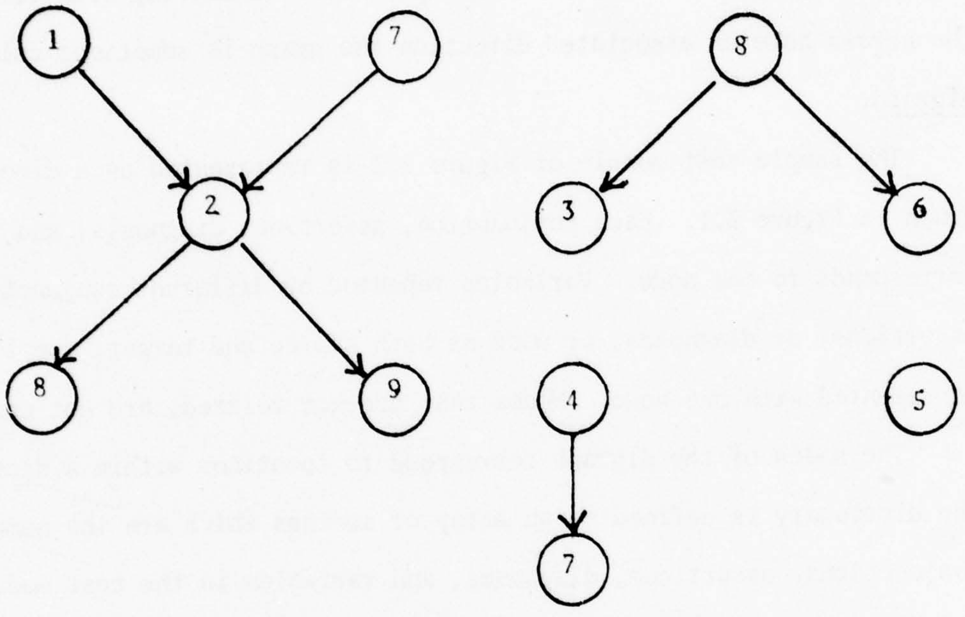
The sample test module of Figure 2.2 is represented as a directed graph in Figure 3.1. Each conjunction, assertion, diagnosis, and variable corresponds to one node. Variables repeated by different conjunctions, assertions, or diagnoses; or used as both source and target, are still represented with one node. Nodes that are not related, are not connected.

The nodes of the digraph correspond to locations within a dictionary. The dictionary is defined as an array of strings which are the names of the conjunctions, assertions, diagnoses, and variables in the test module. This structure stores the digraph node labels and is used to form the rows and columns of the adjacency matrix. It is formed by grouping conjunctions and assertions, diagnoses and variables.

Figure 3.2 shows the construction of the dictionary for the sample test module in Figure 2.2. The dictionary has nine labels prearranged beginning with conjunctions and assertions followed by diagnoses and variables.

Although the directed graph is more comprehensible for humans, it's counterpart, the adjacency matrix, is an equivalent form better suited for digital computation. Formally the adjacency matrix(A) corresponds to the digraph (N,R) of N nodes with one relation R defined as an N x N matrix (i.e. a matrix having an equal number of rows and columns). The relation is expressed below with a "1" in Aij indicating a path from node i to node j:

```
Aij = 1 if (Nj,Nk) is in R; ELSE
Aij = 0
```



DIRECTED GRAPH OF SAMPLE TEST MODULE IN FIGURE 2.2

FIGURE 3.1

1	$S W0001	CONJUNCTIONS
2	$M W0001	
3	$M W0002	ASSERTION
4	4	DIAGNOSES
5	5	
6	6	
7	VAR1	VARIABLES
8	F1	
9	V2	

DICTIONARY OF NODE IN THE DIRECTED GRAPH
IN FIGURE 3.1

FIGURE 3.2

The adjacency matrix for the sample test module of Figure 2.2 is shown in Figure 3.3. It is equivalent to the directed graph in Figure 3.1. such that relations between nodes are preserved. For example, the "1" in row 2 column 8 indicates a directed path from node 2 to node 8. Thus, referring to the dictionary, the path is from the conjunction $M_W0001 to the target variable F1. The next section details the algorithms used to indicate paths between nodes in the adjacency matrix.

STATEMENT TYPE		DICTIONARY	ADJACENCY MATRIX								
			1	2	3	4	5	6	7	8	9
CONJUNCTION	1	$S_W0001	0	1	0	0	0	0	0	0	0
CONJUNCTION	2	$M_W0001	0	0	0	0	0	0	0	1	1
ASSERTION	3	$M_W0002	0	0	0	0	0	0	0	0	0
DIAGNOSES	4	4	0	0	0	0	0	0	1	0	0
DIAGNOSES	5	5	0	0	0	0	0	0	0	0	0
DIAGNOSES	6	6	0	0	0	0	0	0	0	0	0
VARIABLE	7	VAR1	0	1	1	0	0	0	0	0	0
VARIABLE	8	F1	0	0	1	0	0	1	0	0	0
VARIABLE	9	V1	0	0	0	0	0	0	0	0	0

FIGURE 3.3 CONSTRUCTING THE DICTIONARY AND FORMING THE ADJACENCY MATRIX

3.2 EXPRESSING PRECEDENCE RELATIONS IN MATRIX FORM

This section explains the mechanism of linking variables in matrix form to specify the precedence of execution.

Conjunctions and assertions employ source and target variables to help specify logical or arithmetic relationships. A target variable is expressed within the adjacency matrix by placing a "1" at the intersection of the <u>conjunction or assertion</u> row and the target <u>variable column</u>. A source variable is the inverse of the target representation expressed by placing a "1" at the intersection of the source <u>variable row</u> and the <u>conjunction or assertion</u> column. This inverse representation guarantees precedence of target relations before source relations. Figure 3.4 summarizes the process of indicating the above relations.

	MATRIX	ROW	COLUMN	VALUE
TARGET:	ADJACENCY	CONJUNCTION OR ASSERTION	TARGET VARIABLE	1 BINARY
SOURCE:	ADJACENCY	SOURCE VARIABLE	CONJUNCTION OR ASSERTION	1 BINARY

FIGURE 3.4 TARGET AND SOURCE SUMMARY

Figure 3.5 is an adjacency matrix formed from information in Figure 2.2. It illustrates using mnemonics to show explicitly the source and target linkage.

The first entry in the dictionary is conjunction $S_W0001. Matrix entries are not required because the conjunction does not employ source or target variables. The second entry is conjunction $M_W0001 which employs two target variables, F1 and V1; and one source variable, VARI. The target relations are entered in the second row of the adjacency matrix with "1" in the eighth and ninth columns. The source relations are entered with a "1" in the second column of the seventh row. The third entry, assertion $M_W0002, uses two source variables VARI and F1 which are entered in the third column of rows seven and eight.

DICTIONARY ADJACENCY MATRIX

		1	2	3	4	5	6	7	8	9
1	$S_W0001	0	B	0	0	0	0	0	0	0
2	$M_W0001	0	0	0	0	0	0	0	T	T
3	$M_W0002	0	0	0	0	0	0	0	0	0
4	4	0	0	0	0	0	0	R	0	0
5	5	0	0	0	0	0	0	0	0	0
6	6	0	0	0	0	0	0	0	0	0
7	VAR1	0	S	S	0	0	0	0	0	0
8	F1	0	0	S	0	0	P	0	0	0
9	V1	0	0	0	0	0	0	0	0	0
		$S_W0001	$M_W0001	$M_W0002	4	5	6	VAR1	F1	V1

S - SOURCE

T - TARGET

R - OPERATOR RESPONSE

P - OTHER PARAMETERS

B - STIMULI CONJUNCTION EXECUTED BEFORE MEASUREMENT CONJUNCTION

FIGURE 3.5 ADJACENCY MATRIX FORMED FROM TEST MODULE IN FIGURE 7.1

Variables used within diagnoses are represented in a similar form to the source and target variables. An operator response variable is expressed by placing a "1" at the intersection of the diagnosis row and variable column. The other parameters variable is expressed by placing a "1" at the intersection of the variable row and diagnosis column. The inverse representation guarantees precedence of operator responses before other parameters. Literal strings that are enclosed by apostrophes (e.g. 'HZ') have no significance in the logical ordering of statements thus they are not represented in either the dictionary or the adjacency matrix. Figure 3.6 summarizes the method used to indicate variable attributes to diagnosis.

	MATRIX	ROW	COLUMN	VALUE
OPERATOR RESPONSE:	ADJACENCY	DIAGNOSIS	VARIABLE	1 BINARY
OTHER PARAMETERS:	ADJACENCY	VARIABLE	DIAGNOSIS	1 BINARY

FIGURE 3.6 DIAGNOSIS AND VARIABLE SUMMARY

Figure 3.5 also shows operator response and other matrix entries constructed from the sample specifications in Figure 2.2. Mnemonics are used to explicitly differentiate these relations from the previous source and target structures. For example, Diagnosis 4, which employs the variable VARI as an operator response, is expressed in the seventh column of the fourth row. In Diagnoses 5 the other parameters, 'FREQ' is a literal string (denoted by a character string enclosed within apostrophes) rather than a variable. Since the adjacency matrix only links variables, 'FREQ' is not indicated on the sample matrix. Similarly, the literal string 'HZ' of Diagnosis 6 is not present in the adjancency matrix.

However, other parameter variable F1 of Diagnosis 6 is represented in the sixth column of the eighth row.

The path relations introduced thus far directly parallel the waveforms and diagnosis statements. Because NOPAL is non-procedural, an additional relation is necessary to insure that stimuli conjunctions precede measurement conjunctions. Thus the functions which excite the UUT through specified terminals precede the functions which classify UUT's responses. This is achieved simply by inserting a "1" in the measurement conjunction column of the stimuli conjunction row. Thus in the sample test module of Figure 2.2, a "1" is entered in the second column of the first row, linking $S_W0001 to $M_W0001.

The process of constructing the adjacency matrix is summarized in the algorithm of Figure 3.7 The last step of this process evaluates the external variable 'SEQOPT'(initially set when invoking the NOPAL system) to ascertain whether to generate a special purpose Adjacency Matrix Report. This report (Figure 3.8) indicates relations, node labels, and node attributes of the sample test module in Figure 2.1. It summarizes in short form the necessary information employed in intra-test sequencing.

Preliminary error analysis is performed as soon as the above information is accumulated in the adjacency matrix. This analysis detects multiply defined target variables is discussed in Section 4. Section 5 develops a more sophisticated algorithm to detect circular ambiguities. As the reader procedes through the following sections, the power and sophistication of representing test specifications in matrix form will become evident.

ALGORITHM TO CREATE AN ADJACENCY MATRIX

```
LET #W = NUMBER OF CONJUNCTIONS AND ASSERTIONS IN ADJACENCY MATRIX
LET #D = NUMBER OF DIAGNOSES ON ADJACENCY MATRIX
LET #V = NUMBER OF VARIABLES IN ADJACENCY MATRIX
LET N = TOTAL NUMBER OF CONJUNCTIONS, ASSERTIONS, DIAGNOSES VARIABLES
LET DICTIONARY(N) = LIST OF CONJUNCTION, ASSERTION, DIAGNOSIS, AND
    VARIABLE LABELS.  (SIZE DETERMINED BY N).
LET ADJACENCY MATRIX(N,N) = MATRIX SUCH THAT EACH POSITION 'X' CORRESPONDS
    TO THE LABEL IN POSITION 'X' OF THE DICTIONARY.

 1.  GATHER, THEN GROUP ALL CONJUNCTIONS, ASSERTIONS, DIAGNOSES, AND
     VARIABLES INTO DICTIONARY
 2.  COUNT AND SET #W, #D, #V AS NECESSARY
 3.  CALCULATE SIZE OF THE ADJACENCY MATRIX
     N = #W + #D + #V
 4.  ALLOCATE ADJACENCY MATRIX A(N,N)
 5.  SET ADJACENCY MATRIX TO '0'
 6.  PERFORM 7-9 FOR EACH CONJUNCTION AND ASSERTION
 7.  FOR EACH VARIABLE PERFORM 8-9
 8.  IF SOURCE VARIABLE THEN:
        ADJACENCY MATRIX(VARIABLE ROW, CONJUNCTION OR ASSERTION COLUMN) =1
 9.  IF TARGET VARIABLE THEN:
        ADJACENCY MATRIX(CONJUNCTION OR ASSERTION ROW, VARIABLE COLUMN) =1
10.  PERFORM 11-12 FOR EACH DIAGNOSIS (#D)
11.  FOR EACH DIAGNOSIS VARIABLE PERFORM STEP 12-13
```

FIGURE 3.7 ALGORITHM TO CREATE ADJACENCY MATRIX

```
12. IF OPERATOR RESPONSE VARIABLE IN DIAGNOSIS THEN:
        ADJACENCY MATRIX (DIAGNOSIS NODE, VARIABLE NODE) = 1
13. IF OTHER PARAMETER VARIABLE IN DIAGNOSIS THEN
        ADJACENCY MATRIX (VARIABLE NODE, DIAGNOSIS NODE) = 1
14. IF THERE EXISTS A STIMULI AND MEASUREMENT CONJUNCTION WITHIN TEST MODULE
        THEN ADJACENCY MATRIX (STIMULI CONJ. NODE, MEASUREMENT CONJ. NODE)=1
15. IF 'SEQOPT' = 1 THEN PRINT ADJACENCY MATRIX REPORT
16. END
```

FIGURE 3.7 ALGORITHM TO CREATE ADJACENCY MATRIX

(continued)

```
                              INTRA MODULE SEQUENCING SYTEST0003
                              ANALYSIS OF THE ADJACENCY MATRIX

                                             1 2 3 4 5 6 7 8 9
                                             - - - - - - - - -
 1  $S W0001     CONJUNCTION                 0 1 0 0 0 0 0 0 0
 2  $M_W0001     CONJUNCTION                 0 0 0 0 0 0 0 1 1
 3  $M_W0002     ASSERTION                   0 0 0 0 0 0 0 0 0
 4  4            DIAGNOSES                   0 0 0 0 0 0 1 0 0
 5  5            DIAGNOSES                   0 0 0 0 0 0 0 0 0
 6  6            DIAGNOSES                   0 0 0 0 0 0 0 0 0
 7  VAR1         VARIABLE                    0 1 1 0 0 0 0 0 0
 8  F1           VARIABLE                    0 0 1 0 0 1 0 0 0
 9  V1           VARIABLE                    0 0 0 0 0 0 0 0 0
```

ADJACENCY REPORT

FIGURE 3.8

4. PRELIMINARY ANALYSIS OF THE ADJACENCY MATRIX

After entering the known precedence relations into the adjacency matrix an analysis is performed to guarantee that the test module is error free. This section corresponds to the second box of the center partition in Figure 1.2. Methods verifying target variables correct usage are discussed. Specifically four types or ambiguities or inconsistencies are presented: 1) excess target variables per assertion, 2) incorrect target variable expression, 3) wrong arithmetic operator in an assertion, and 4) target variable defined more than once.

Section 4.1 presents an overview of the analysis, and section 4.2 details the algorithm.

4.1 OVERVIEW OF THE ANALYSIS

This section begins the evaluation of conjunctions, assertions, and diagnosis variables. To clarify the forthcoming discussion, a new sample test module is presented in Figure 4.1. This test module is composed of two conjunctions, three assertions, and one diagnosis. The logic to invoke the diagnoses has been eliminated since it is not a contributing factor in intra-test sequencing. Although the test module may appear well formed, it posses multiple errors that would prevent sequencing and final code generation.

The methods used to ascertain the correctness of a test module are dependent on the adjacency matrix. Because it contains the representation for all conjunctions, assertions, diagnoses, and variables, it reduces the necessity of retrieving the original source strings from the associative memory.

```
TEST ONE;
    STIMULI $S_ONE(ONE);
        CONJUNCTION A3:
            (J22,GND) = SINE_D (Y,VOLT, Z, Z, X SEC)
                TARGET:  Y, Z
                SOURCE:  X;
    MEASUREMENT $M_ONE(ONE);
    CONJUNCTION A2:  (  J24_3, GND)   =  CONS_T(27.5 VOLT));
    ASSERTION A4($M_ONE):
        X = Y
            TARGET:  Y,Z
            SOURCE:  X;
    ASSERTION A5($M_ONE):
        X-1 = Y
            TARGET:  X
            SOURCE:  Y;
    ASSERTION A6($M_ONE):
        Y < X
            TARGET:  Y
            SOURCE:  X;
    DIAGNOSE D1:
        OPERATOR RESPONSE=(X)
        TYPE = D;
```

FIGURE 4.1 SAMPLE TEST MODULE WITH UNDETECTED ERRORS

Target variables and operator responses have special significance in the preliminary analysis. To achieve successful sequencing of the test module, the variables must explicitly adhere to the conventions established in the NOPAL Language Reference Manual. Rather than examining each statement in the test module, the analysis can be expedited by using a special partition within the adjacency matrix that contains all target and operator responses. Formally, the partition consists of the columns that refer to variables and the rows that refer to conjunctions assertions and diagnoses. The commonality of variables in this partition is that the target variable is evaluated after the conjunction or assertion is executed while the operator response of a diagnosis is set after the diagnostic.

The adjacency matrix for the sample test module is presented in Figure 4.2. The relevant partition to the discussion is surrounded by a heavy broken line. It contains all target and operator responses in the test module. An additional column labelled "MT" is inserted to the right of column nine. The contents of this vector are the total number of target variables per conjunction or assertion. It begins in row 1 and terminates in row 5. Below row nine, an additional row labelled "MD" is inserted. It contains the total number of times each variable is defined to follow the conjunction, assertion, or diagnoses. Thus it is calculated by adding the content of each column, **and expanding the partition to include diagnoses. Enhanced processing time is achieved by** using the matrix as well as the additional row and column.

The next section describes the algorithm that evaluates the relations in the aforementioned partition.

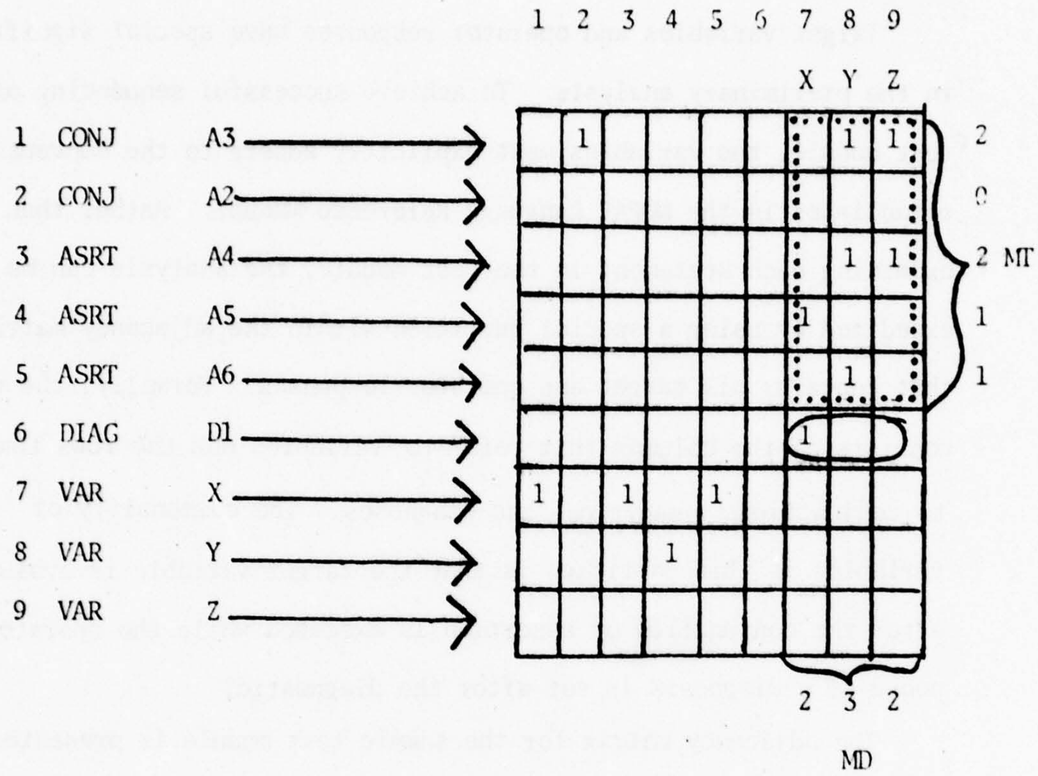


FIGURE 4.2

ADJACENCY MATRIX FOR SAMPLE TEST MODULE - FIGURE 4.1

4.2 ALGORITHM PERFORMING ANALYSIS

Inconsistencies discussed in this section are grouped in two categories: 1) assertions that employ target variables and 2) conjunctions, assertions, and diagnoses that employ the same target variable and/or operator response. A summary of the user messages is shown in Figure 4.3. The overall classification of the message is on the left side of the diagram, with the actual text on the right side. Each error or warning message states the test module, the statement, the associated variable, and the ambiguity. Abbreviations are not used to avoid confusing the inexperienced user.

Most of this analysis concerns the special conventions used with assertions that employ target variables. Of special concern is that each assertion may use only one target variable. The verification of this is implemented by searching the "MT" column in the sample test module. If any sum is greater than one, the process continues to check whether the node in the dictionary is an assertion. If this situation does occur, the first error message of Figure 4.3 is issued. An example of this type of error is shown with assertion A4 (Figure 4.1) having two target variable, y and z. The "2" in the first position of MT is not significant as the dictionary node indicates that it is a conjunction.

Another convention pertaining to assertions that employ target variables is that the variable must precede the equal sign. Expansions or modifications by performing multiplication, division, subtraction, or addition are not allowed before the equal sign. This convention is consistent with the standard mathematical notation of placing the result of a computation of the variable preceding the equal sign. The

TYPE	MESSAGE
1 *ERROR (AMBIGUITY):	IN ASSERTION________OF TEST____________, THERE ARE TWO OR MORE TARGET VARIABLES ____, _______, _______.
2 *ERROR (AMBIGUITY):	EXPRESSION________PRECEDING THE'=' IN ASSERTION__________OF TEST________DOES NOT MATCH THE TARGET VARIABLE____________.
3. *WARNING (INCONSISTENCY):	IN ASSERTION___________OF TEST_________, A VARIABLE IS DECLARED AS TARGET; BUT THE RELATION OPERATOR IS NOT AN'='. REPLACED BY AN EQUAL SIGN.
4. *WARNING (POSSIBLE AMBIGUITY):	VARIABLE________OF TEST__________ IS DEFINED MORE THAN ONCE IN ____, _____, _______.

FIGURE 4.3 SUMMARY OF ERROR/WARNING MESSAGES

verification is implemented by first searching the "MT" column of the adjacency matrix to ascertain whether the waveform employs a target variable. If any sum in this column is greater than one, the process continues to check whether the node in the dictionary is an assertion. Then the target variable indicated within the isolated partition is compared to the text retrieved from the associative memory which precedes the equal sign. If both expressions are not equivalent, the second user message in Figure 4.3 is issued. An example of this type of ambiguity is shown in assertions A5 and A6. (Figure 4.1) Target variables Y or Z used in A4 are different than X, the actual expression which precedes the equal sign. Also target variable X of Assertion A5 is different than the actual expression "x-1" which precedes the equal sign.

The equal sign is the sole operator permitted when assertions employ target variables. The implementation strategy is similar to the assertion categories already discussed. By evaluating "MT" and the dictionary, assertions are located that employ target variables. Once these assertions are known, the operator is retrieved from the associative memory. If it is not an equal sign, the third user message is issued. Concurrently, the operator is changed to an equal sign in the associative memory. For example assertion A6 exhibits this characteristic by using a less than operator instead of an equal sign.

Should the same variable be the target of a conjunction of assertion, and also the operator response of a diagnose, there exists a possibility that the user has made an error by multipley defining the variable. The implementation is different from the preceding three as the partition (Figure 4.2) is expanded to include operator responses. This is accomplished by including the diagnosis rows as shown in the circle surrounding columns 7,8,9 of row 6.

Once the partition is enlarged, the sum of each column is stored in an additional row, "MD", located at the bottom of the adjacency matrix. The contents of "MD" indicate the total number of times each variable is defined as target to a conjunction, target to an assertion, and the operator response of a diagnosis. If any sum is greater than one, the fourth user message is issued. The sample test module (Figure 4.1) displays three examples of this ambiguity. They are:

1) variable X defined in assertion A5 and diagnosis D1.
2) variable Y defined in assertions A4, A6 and conjunction A3.
3) variable Z defined in assertion A4 and conjunction A3.

The process of performing the analysis discussed in this section is summarized in the algorithm of Figure 4.4. Most of the analyses are performed simply by going up, down, and across the adjacency matrix searching for "1's" indicating predetermined relations.

Even though a procedure successfully passes the preliminary analysis covered in this section and in syntax analysis, it does not necessarily guarantee that there are no hidden or inferred ambiguities. For example the sample test module possesses a circular definition that cannot be sequenced. Discovering its existence is a non-trivial task whose complexity may be seen by attempting to manually locate it. Just as it is a difficult process in a short test module of only six lines, imagine the difficulty when using a test module exceeding fifty lines. The complexity of detecting these inconsistencies manually led to the development of an automatic process of checking all test modules for this type of inconsistency.

SUMMARY ALGORITHM

PRELIMINARY ANALYSIS OF THE ADJACENCY MATRIX

```
LET #W = NUMBER OF CONJUNCTIONS AND ASSERTIONS IN ADJACENCY MATRIX
LET #D = NUMBER OF DIAGNOSES IN ADJACENCY MATRIX
LET #V = NUMBER OF VARIABLES IN ADJACENCY MATRIX
LET #N = TOTAL NUMBER OF CONJUNCTIONS, ASSERTIONS, DIAGNOSES, VARIABLES
LET HOM1 = #N + 1
LET DICTIONARY (N) = LIST OF CONJUNCTION, ASSERTION, DIAGNOSIS,
    AND VARIABLE LABELS. (SIZE DETERMINED BY N)
LET ADJACENCY MATRIX(N,N) = MATRIX SUCH THAT EACH POSITION CORRESPONDS
    TO THE LABEL IN POSITION X OF THE DICTIONARY.

1.  /* MORE THAN 1 TARGET VARIABLE PER ASSERTION #
2.  PERFORM 3-6 FOR EACH CONJUNCTION AND ASSERTION (I=1 to #W)
3.  #E=0
4.  IF DICTIONARY(I) IS AN ASSERTION THEN
        PERFORM 5 FOR EACH POSSIBLE TARGET VARIABLE (J=HOM1 TO N)
5.  IF ADJACENCY MATRIX(I,J) = '1' THEN #E = #E + 1
6.  IF #E > 1 THEN PRINT ERROR #1
7.  /* TARGET VARIABLE MUST PRECEDE ARITH OPERATOR */
8.  PERFORM 9-12 FOR EACH CONJUNCTION AND ASSERTION (I=1 TO #W)
9.  IF DICTIONARY (I) IS AN ASSERTION THEN
        PERFORM 10-12 FOR EACH POSSIBLE TARGET VARIABLE (J=HOM1 TO N)
10.     IF ADJACENCY MATRIX (I,J) = '1' THEN
            PERFORM 11-12
```

FIGURE 4.4

```
11.          RETRIEVE TEXT PRECEDING OPERATOR
12.          IF TEXT 7 = DICTIONARY (J) THEN PRINT ERROR #2
13.   /* ASSERTIONS ARITHMETIC OPERATOR */
14.   PERFORM 15-18 FOR EACH CONJUNCTION AND ASSERTION (I=1 TO #W);
15.   IF DICTIONARY(I) IS AN ASSERTION THEN
             PERFORM 16-18
16.   #E = 0
17.   RETRIEVE OPERATOR FROM THE ASSOCIATIVE MEMORY
18.   IF OPERATOR IS NOT EQUAL TO '=' THEN PRINT ERROR #3
         AND REPLACE ASSERTION'S OPERATOR IN THE ASSOCIATIVE MEMORY WITH '='.
19.   /* MULTIPLEY DEFINED TARGET VARIABLES */
20.   PERFORM 21-24 FOR EACH VARIABLE (I= HOM1 TO N)
21.   #E = 0
22.       PERFORM 23  FOR EACH CONJUNCTION, ASSERTION, DIAGNOSIS (J=1 TO
          #W + #D)
23.       IF ADJACENCY MATRIX(J,I) = '1' THEN #E = #E + 1
24.   IF #E > 1 THEN PRINT ERROR #4
25.   END
```

FIGURE 4.4

5. PATH MATRIX

This section discusses the process of detecting circular definitions that might exist in the digraph. It corresponds to the third box within the center partition of the NOPAL processor diagram (Figure 1.2). The implementation is achieved by constructing a 'path' matrix to reveal direct and implied paths. Transitive or circular definitions are easily isolated in this matrix. Upon detection, additional processing is subsequently performed to isolate their exact nodes and report them to the user. Section 5.1 discusses an overview of the process while section 5.2 details the algorithm which constructs the path matrix.

5.1 OVERVIEW OF DETECTING CIRCULAR DEFINITIONS

As the number of digraph nodes increase, there exists a greater possibility of generating a cycle. A cycle is defined as a sequence of inter-connected nodes such that the final node and the beginning node are the same. Simply it is a closed loop, or endless sequence of test statements. This ambiguity must be corrected before proceeding to the sequencing phase of the NOPAL processor.

A sample digraph containing circular ambiguity is shown in Figure 5.1. The digraph nodes correspond to dictionary labels "1", "2", "3", "4", "5". The most obvious cycle present exists between nodes "1" and "5". Node "1" has a direct path to node "5"; similarly node "5" has a direct path to node "1". Therefore node "1" must precede node "5" AND node "5" must precede node "1". This ambiguity is one of many present within the digraph that must be detected before proceeding to the intra test sequencing.

If all test modules contained less than ten positions, then detecting closed loops would be trivial. By generating a pictorial graph of the test module, the user could then examine it to veryify it is cycle free.

However as the size of test modules grows, exceeding fifty positions, the manual process becomes increasingly complex. Performing this function manually is inappropriate for a general purpose system which promotes the simplification of specifying test situations. Thus an automatic algorithm was incorporated to detect the presence of circular definitions.

The adjacency matrix for the digraph (Figure 5.1) is presented in Figure 5.2. Each path is a direct link between two digraph nodes. For example the path emanating from node "4" and terminating at node "5" is indicated in the fifth column of the fourth row. Indirect paths such as the one beginning at node "4" continuing through nodes "2", "1", "3", and ultimately terminating at node "4" are not shown in this matrix.

An efficient method to detect circular definitions involves creating a matrix that exhibits all paths. A <u>path</u> matrix if formed as a transformation of the adjacency matrix that accomplishes this goal. Formally the path matrix indicates the existence of paths regardless of length from node i to node j. It is sometimes referred to as a 'reachability' matrix due to its attribute of displaying a summary of nodes which may be reached from other nodes.

The path matrix for the sample digraph is presented in Figure 5.3. Every entry has a "1" showing that any node may reach any other node including itself. For example node "4" may reach nodes "1", "2", "3", "4". In essence this matrix reports the existence of all paths, direct or indirect.

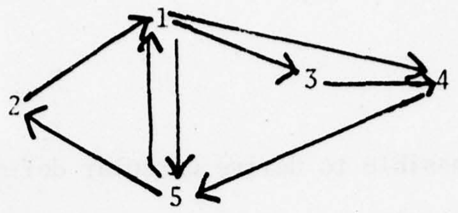


FIGURE 5.1

SAMPLE DIGRAPH WITH CYCLE

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 0 & 1 & 1 & 1 \\ 2 & 1 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 1 & 1 \\ 4 & 0 & 0 & 0 & 0 & 1 \\ 5 & 1 & 1 & 0 & 0 & 0 \end{array}$$

ADJACENCY MATRIX FOR SAMPLE DIGRAPH

FIGURE 5.2

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & 1 & 1 & 1 & 1 \\ 3 & 1 & 1 & 1 & 1 & 1 \\ 4 & 1 & 1 & 1 & 1 & 1 \\ 5 & 1 & 1 & 1 & 1 & 1 \end{array}$$

PATH MATRIX FOR SAMPLE DIGRAPH

FIGURE 5.3

'1' ON DIAGONAL INDICATES CYCLES IN DIGRAPH

Although it is possible to derive circular definitions directly from the adjacency matrix, it is not a computationally efficient approach. Once the path matrix is created, the presence of one or more cycles is detected by searching for a "1" on the diagonal. If there are no cycles in the graph, the system procedes to the next phase, precedence determination. Otherwise, the nodes that constitute the closed loop are determined using the 'cycle' procedure shown in Section 6.

The following section explains the transformation performed to produce the path matrix.

5.2 CONSTRUCTING THE PATH MATRIX

The path matrix (P) consists of ones and zeroes with a "1" in row i and column j if there is a path from node i to node j. The procedure used to construct this matrix is adapted from Warshall's algorithm [Warshall, 1962] summarized in Figure 5.4.

The technique begins by copying the adjacency matrix into an empty path matrix. The columns of the path matrix are then traversed searching for "1's". For each "1" in Pij, a path exists from node i to node j. Therefore any path from node j is also common to node i. Each path (k) from row j is copied to the k column of row i.

For example the adjacency matrix of Figure 5.2 shows node "1" having paths to nodes "3", "4", "5". The sole path in node "2" is to node "1". Since node "1" can be reached from node "2", all paths from node "1" can also be indirectly reached from node "2". The algorithm continues by placing "1's" in column "3", "4", "5" of row 2.

The constructed path matrix reveals the presence of cycles by exhibiting "1's" on any diagonal. The next section (6) explains the process of enumerating the distinct nodes that constitute the cycle.

```
P = path matrix
A = adjacency matrix
n = number of nodes in digraph

1.  Let P = A (for all ij)
2.  Set j = 1
3.  Set i = 1
4.  IF Pij = "1" then Pik = Pik  Pjk (for all k=1 to n)
5.  Set i = i+1; if i = n, THEN go to 4
6.  Set j = j+1; If j = n, then go to 3; else return;
```

ALGORITHM TO CONSTRUCT PATH MATRIX
FIGURE 5.4

6. ENUMERATING CIRCULAR DEFINITIONS

This section discusses the procedure which locates the exact statements constituting circular definitions. The implementation is associated to the fourth box within the center partition of the NOPAL processor diagram (Figure 1.2). It is invoked, if and only if, cycles are detected on the diagonal of the path matrix. In essence, the algorithm provides a user aid for debugging inconsistencies in the test module. The discussion begins with a general overview of the technique employed followed by an illustration.

6.1 BUILDING TREES TO LOCATE CIRCULAR DEFINITIONS

There are three parameters used to isolate circular definitions within the digraph. They are the adjacency matrix (A), the path matrix (P), and the number of nodes (n) in the digraph. The algorithm finds cycles by the principle that node i is in a cycle with node k, if A_{ik} $P_{ki} = 1$; i.e. there is a path from node i to node k and a path from node k back to the node i. Formally, the extension of digraph paths adequately traces circular definitions.

Extended paths isolating circular definitions are referred to as trees. A tree is a set of connected nodes that begin at a predetermined root or top node, and continue to a terminal or final node. At each interval, as the tree increases in length, the terminal node is compared to the root node. If they are equivalent the tree is circularly defined. Else the tree continues to grow until the end of the path is reached or the tree length exceeds the number of nodes in the digraph.

Adapted from [Berztiss, 1971] the tree generating algorithm is summarized in Figure 6.1. It commences by building the tree beginning with node "1" where each extension is:

$$A_{1i} \times P_{i1} = "1" \quad (i=1 \text{ to } n)$$

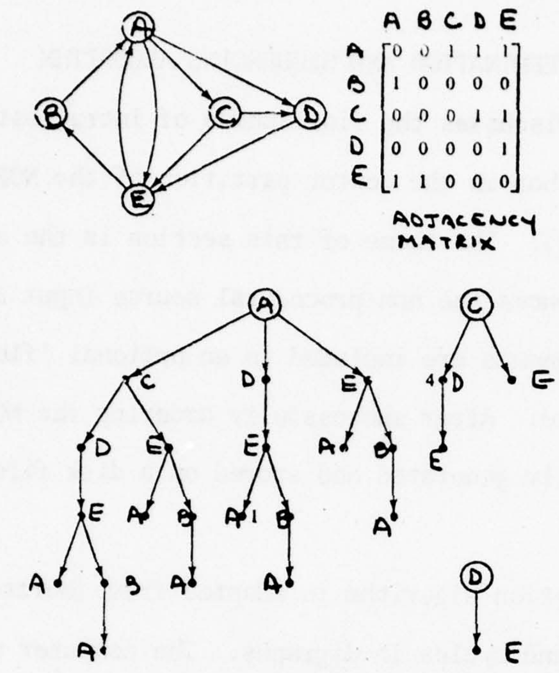
Figure 6.2 shows a sample digraph, its' adjacency matrix, and the constructed trees. The algorithm deletes past root columns and rows to expedite processing. Figure 6.3 illustrates the output produced from the "cycles" algorithm invoked to evaluate the sample digraph. The error message explicitly defines ambiguous nodes.

There are several methods which might be used to correct the circular inconsistency. The user should initially begin by reviewing the NOPAL Test Specification Report to verify that the statements and variables correspond to the intended source file. The next logical procedure is to scan the Cross Reference and Attribute Report for the existence of slightly different labels, which may be the result of typographical error. The Adjacency Matrix Report is useful because it reproduces the system interpretation of the test module. Warning and error messages produced during the preliminary analysis of the adjacency matrix may be the best clue to those variables incorrectly defined. The above methods combined with the output from "cycles" provide extensive diagnostic assistance.

```
                                    Let A = Adjacency Matrix
                                    Let P = Path Matrix
                                    Let N = Number of notes in Digraph
                                    Let Root = Beginning node of tree
                                    Let Level = Number of nodes in tree
                                    Let Path(k) = Actual nodes in tree

Algorithm CYCLES:  Cycle Enumeration

Step 1.  Root = 1
Step 2.  (initiate tree; steps 2 to 6):
      Set REACHJ  (k) = Root (for k=Root to n)
Step 3.  Set USED (k) = 0 (for k=Root to n)
Step 4.  Set level=1.
Step 5.  PATH (1)=Root.
Step 6.  Set i=Root.

Step 7.  (Test if current path can be extended with nodes in a cycle:
          Steps 7-11):
     IF REACHJ (i) > n then go to Step 12.
Step 8.  Set j= REACHJ  (i).
Step 9.  If A(i,j)*P(j,Root)=1 and ¬USED (j) then go to Step 18.
Step 10. Set j=j+1.
Step 11. If j < =n then go to Step 9.

Step 12. (Backtrack in tree, resetting REACHJ and USED ;
      Steps 12 through 17):
      Set REACHJ (i) = Root.
Step 13.  Set USED (i) = 0.
Step 14.  Set level=level-1.
Step 15.  If level=0 then go to Step 26.
Step 16.  Set i = PATH (level).
Step 17.  Go to Step 7.

Step 18.  (Extend path; Steps 18 through 23):
      Set USED (j) = 1.
Step 19.  Set REACHJ (i) = j+1
Step 20.  Set level=level - 1.
Step 21.  Set PATH (level) = j.
Step 22.  Set i=j.
Step 23.  If j ¬=Root then go to Step 7.

Step 24.  (Print Cyclic Path):
      Print PATH (k), k = 1 to level (message #7).
Step 25.  Go to Step 13.
Step 26.  Set Root=Root+1.
Step 27.  If Root <= n then go to Step 2.
Step 28.  Return
```

"CYCLES ALGORITHM"

FIGURE 6.1



SAMPLE DIGRAPH, ADJACENCY MATRIX, AND TREES

FIGURE 6.2

```
ERROR (Circular definition): The following group of items in test above
are circularly defined:

        A,C,D,E,A
        A,C,D,E,B,A
        A,C,E,A
        A,C,E,B,A
        A,D,E,A
        A,D,E,B,A
        A,E,A
        A,E,B,A
```

SAMPLE OUTPUT FROM CYCLE ENUMERATING PROCEDURE ANALYZING DIGRAPH OF FIGURE 6.2

FIGURE 6.3

7. PRECEDENCE DETERMINATION AND SEQUENCING ALGORITHM

This section discusses the final phase of intra test sequencing shown by the fifth box in the center partition of the NOPAL processor diagram (Figure 1.2). The theme of this section is the algorithm which automatically sequences the non-procedural source input statements. The reordered statements are included in an optional 'flowchart' report for user convenience. After successfully ordering the NOPAL statements, code is automatically generated and stored on a disk file by the processor.

The implementation algorithm is adapted from [Berztiss, 1971] analysis of paths and cycles in digraphs. The computer program is an adaptation of a program written by Adam Rin in his Ph.D. dissertation [Rin, 1976]. Rins implementation automatically generates a business applications program from non-procedural source specification statements. Although the application is functionally different than devising techniques to generate programs for automatic test equipment; the sequencing process for non-procedural input is fundamentally equivalent.

The following sections present an overview of the sequencing algorithm followed by an example including the optional flowchart report.

7.1 OVERVIEW OF PRECEDENCE DETERMINATION

The comprehensive design of the adjacency matrix insures that it contains the necessary information for intra-test sequencing. After complex error analysis verifies the validity of the digraph, the nodes are ranked according to precedence and reordered according to their rank.

The easiest method to implement the sequencing strategy is through matrix multiplication. For example, the adjacency matrix A, displays paths of length 1 from i to j. A^2 displays paths of length 2. Formally A^j displays the paths of length j. At each stage of the matrix multiplication, the column of the current rank set contains all zeros; having no predecessor with the current length path. Employing a well formed digraph, the algorithm terminates when all entries in A are zero requiring at most n stages $(j = n)$.

Although the above approach is straight-forward, it is not the most efficient implementation because the matrix multiplication requires n^4 steps. A much quicker algorithm to sequence the digraph nodes is given in algorithm "preceed" (Figure 7.1). This algorithm analyzes the original matrix in n^2 steps without performing multiplications.

The algorithm works by first finding all the nodes of rank 0; i.e. all the nodes which do not have precedents (Step 2). This is simply all the nodes which have all zeros in their column (in Step 2, these are all column nodes j that are put in set D(0); the "i" in the condition are the row entries in each such column j). These nodes become the elements of rank set D(0), and the rank of all such

Algorithm (PRECEED): Precedence Determination

the following symbols are used:

A	The input n x n adjacency matrix (row and column for each node)
i	row index for A
j	column index for A

D a vector of "rank sets"; each rank set (element of the vector) consists of a set of nodes at that rank.

l rank counter; index to D (i.e. in the algorithm, D(l) is the set of nodes of rank l; D(l-1) is the set of nodes of rank l-2 etc.)

n the number of nodes; also the number of rows and columns of A; also the number of elements in vectors R and 0.

P is set successively to each node in the previous rank set, D(l-1); indexes row of A

q is set successively to each node in the current rank set D(l); indexes column of A; also indexes R

R the "rank vector" that is produced (has n elements); the index to R is a node number; the value of each element of R gives the rank of that node; e.g. R(q) gives the rank of node q.

0 the "order vector" produced (has n elements); the indices to 0 are the sequence or step numbers (1,2,3,...); the value at each element of 0 is the node number to be executed at that position.

ALGORITHM (PRECEED) PRECEDENCE DETERMINATION

FIGURE 7.1

STEP	ALGORITHM	EXPLANATION
1	Initialize R to all zeros	Initially Rank vector all 0.
2	$D_0 = \{ j \mid \forall_i \ (A_{ij} = 0) \}$	Nodes of rank 0 consist of all those which have no precedents
	If $D_0 = \emptyset$ then go to 9	i.e. all 0 column
3	$l \leftarrow 0$	Index for rank set initially 0
4	$l \leftarrow l+1$ If $l = n$, go to 9	
5	$D_\ell = \bigcup_{p \in D_{\ell-1}} \{ q \mid A[p,q] = 1 \}$	Next rank set consists of all those nodes which depend on something in the previous rank
6	$R_q \leftarrow \ell \ (\forall q \in D_\ell)$	All nodes in current rank set are ranked with level l
7	If $D_\ell \neq \emptyset$ then go to 4 otherwise go to step 8	If there are still nodes in current rank set, go back to find next rank set
8	Set Order vector to Rearranged nodes in Rank ascending order (normal exit)	Nodes are now ranked; simply rearrange nodes in rank order
9	There exists at least 1 one cycle somewhere in the digraph	This is because the algorithm has gone thru n rank sets and dependencies still exist on last one

ALGORITHM PRECEED: PRECEDENCE DETERMINATION
FIGURE 7.1

nodes to set to 0.

Secondly, the nodes of rank 1 are direct descendants of nodes in rank 0; similiarly the nodes of rank 2 depend on nodes in rank 1 (possibly updating the previous rank of some nodes); and so forth (Steps 3-6).

At each stage, the algorithm has to check the rows of the previous set of nodes for direct descendants (Step 5). After the nodes have thereby been partitioned into <u>rank sets</u>, the order of execution of the nodes is simply a re-arrangement of the nodes according to their <u>rank</u> (Step 8). The result of this algorithm is an <u>order vector</u> 0, where 0(i) is the node to be executed at step i.

The algorithm terminates when either all nodes have been ordered or, theoretically if a cycle exists in a network. The latter is impossible, however, because any cycle would have been detected in the earlier cycle detection and enumeration algorithm, and the processor would not have reached this point.

To best illustrate the use of this algorithm, it is applied to the adjacency matrix of Figure 7.2 which corresponds to the test module in Figure 2.2. Beneath the adjacency matrix (Figure 7.3) is the rank set with 5 partitions of sequenced nodes. Explicitly nodes 1,4,5 of D(0) must precede node 7 of D(1). Similarly node 7 must precede node 2 in D(2). Node 2 must precede nodes 8 and 9 in D(3). Finally nodes 3 and 6 in D(4) are the last nodes to be executed.

The existence of multiple nodes per partition such as in D(0), D(3), D(4) indicate the possibility of executing the nodes in parallel.

```
                                      INTRA MODULE SEQUENCING SYSTEST0003
                                      ANALYSIS OF THE ADJACENCY MATRIX

                                                1 2 3 4 5 6 7 8 9
                                                - - - - - - - - -
1    $S_W0001     CONJUNCTION                   0 1 0 0 0 0 0 0 0
2    $M_W0001     CONJUNCTION                   0 0 0 0 0 0 0 1 1
3    $M_W0002     ASSERTION                     0 0 0 0 0 0 0 0 0
4    4            DIAGNOSES                     0 0 0 0 0 0 1 0 0
5    5            DIAGNOSES                     0 0 0 0 0 0 0 0 0
6    6            DIAGNOSES                     0 0 0 0 0 0 0 0 0
7    VAR1         VARIABLE                      0 1 1 0 0 0 0 0 0
8    F1           VARIABLE                      0 0 1 0 0 1 0 0 0
9    V1           VARIABLE                      0 0 0 0 0 0 0 0 0
```

ADJACENCY MATRIX FOR TEST MODULE (FIGURE 2.2)
FIGURE 7.2

RANK SET		SEQUENCED NODE LABELS
D(0)	1 →	$S_W0001
	4 →	4
	5 →	5
D(1)	7 →	VAR1
D(2)	2 →	$M_W0001
D(3)	8 →	F1
	9 →	V1
D(4)	3 →	$M_W0002
	6 →	6

RANK SET FOR SAMPLE-DIGRAPH (FIGURE 2.2)
FIGURE 7.3

The present system uses a single central processing unit; thus multiple nodes per partition are executed sequentially. As the research into parallel processing continues, future systems may better utilize this characteristic.

A reorganization of the statements is produced in the Intra Test Sequencing Report of Figure 7.4. This report, which resembles a flow-chart, lists the reordered statements within a particular test module. Statement and variable names with the actual rank vector are included for completeness. The actual logic for generating diagnoses is not included as it is not employed during the sequencing of statements. The variables are classified as global, local, source and target. The report which is generated by setting "SEQOPT" to 1 or 2 when invoking the NOPAL processor, is a comprehensive summary of the sequenced test module.

```
                    SEQUENCE OF PROCESSING FOR TEST SYSTEST0003

ORDER
VECT   ORDER    RANK    NAME      TYPE               TEST
INDEX  VECTOR

 1      1        0     $S_W0001   CONJUNCTION     (  J24_B, GND  =  CONST_S(27.5 VOLT ));

 2      4        0     4          DIAGNOSES       TYPE = #5, TIME = 0.00000E+00, RESPONSE=(VAR1);

 3      5        0     5          DIAGNOSES       AFFECTED COMPONENTS = FREQ_TOL(STD_5MHZ_FRE);

                                                  OTHER PARAMETERS=('FREQ'),TYPE = #6

 4      7        1     VAR1       VARIABLE        LOCAL

 5      2        2     $M_W0001   CONJUNCTION     (  J22, GND  = SINE_D(V1 VOLT ,F1 HZ .VAR1 SEC ))
                                                  TARGET:  F1,  V1
                                                  SOURCE:  VAR1;

 6      8        3     F1         VARIABLE        LOCAL

 7      9        3     V1         VARIABLE        GLOBAL / TARGET /

 8      3        4     $M_W0002   ASSERTION       IF VAR1=60 THEN F1 =  5*1E+06 +- 60
                                                  ELSE F1 = 5*1E+06 +- 2.5
                                                  SOURCE:  VAR1,  F1;

 9      6        4     6          DIAGNOSES       OTHER PARAMETERS=('HZ', F1), TYPE = D;
```

SEQUENCE OF PROCESSING REPORT FOR TEST MODULE IN FIGURE 2.2
FIGURE 7.4.

BIBLIOGRAPHY

1. Arinc, "Abbreviated Test Language For Avionics Systems (ATLAS)," Specification 416, 1972. Aeronauntical Radio, Inc., 2551 Riva Road, Annapolis, Maryland 21401.

2. Berztiss, A.T., "Data Structures, Theory and Practice," Academic Press, 1971, New York.

3. Y. Chang, "Automatic Test Program Generation," Dissertation in preparation, Department of Computer and Information Sciences, University of Pennsylvania, Philadelphia, Pennsylvania, 19174.

4. H. Che and Y. Chang, "The NOPAL Language: "Specifications and User Manual," Moore School Report 76-04, University of Pennsylvania, Philadelphia, Pennsylvania, 19174, August 1976.

5. Digitest, "Logic Automated Stimulus And Response (LASAR)," Training Manual, POB 10611, Dallas, Texas 75207.

6. Morris Elecion, "Automatic Testing: Quality Raiser A Dollar Saver," IEEE Spectrum, August 1974, pp. 38-43.

7. Frankford Arsenal, "Operational Performance Analysis Language (OPAL). Definition Of, Syntax Of, and Semantics Of," Proposed MIL-STD-1462, Change 2, Philadelphia, Pennsylvania, 19137, September 1976.

8. F. Linguitti, "Testing Design Methodology: A Case Study," M.S. Thesis. Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania 19174.

9. N.S. Prywes, "Automatic Computer Program Generation For Automatic Testing Systems," Report FCF-3-75, Frankford Arsenal, Philadelphia, Pennsylvania, 19137, January 1975.

10. N. Adam Rin, "Automatic Generation of Business Data Processing Programs For A Non-Procedural Language," Ph.D. Dissertation, 1976, Department of Computer and Information Sciences, University of Pennsylvania, Philadelphia, Pa., 19174.

11. Cihan Tinaztepe and R. Berkowitz, "Automatic Test Program Generation For Automatic Testing Systems," Progress Report, Prepared for Frankford Arsenal Under Contract DAAA-25-75-0650, University of Pennsylvania, Philadelphia, Pennsylvania, 19174, March 1976.

12. Cihan Tinaztepe, "Automatic Test Design," Ph.D. Dissertation in Preparation, Dept. of Computer and Information Sciences, University of Pennsylvania, Philadelphia, Pennsylvania, 19174.

BIBLIOGRAPHY (continued)

13. U.S. Bureau of Labor Statistics, Handbook of Labor Statistics 1975, Washington, D.C., Government Printing Office (B1S Bulletin 1965).

14. J. Warfield, "Binary Matrices In Systems Modelling," IEEE Transactions On Systems, Man and Cybernetics, Vol, SMC-3, No. 5, Sept. 1973.

15. S. Warshall, "A Theorem On Boolean Matrices," JACM 9, 11-12 (1962).

ACKNOWLEDGEMENTS

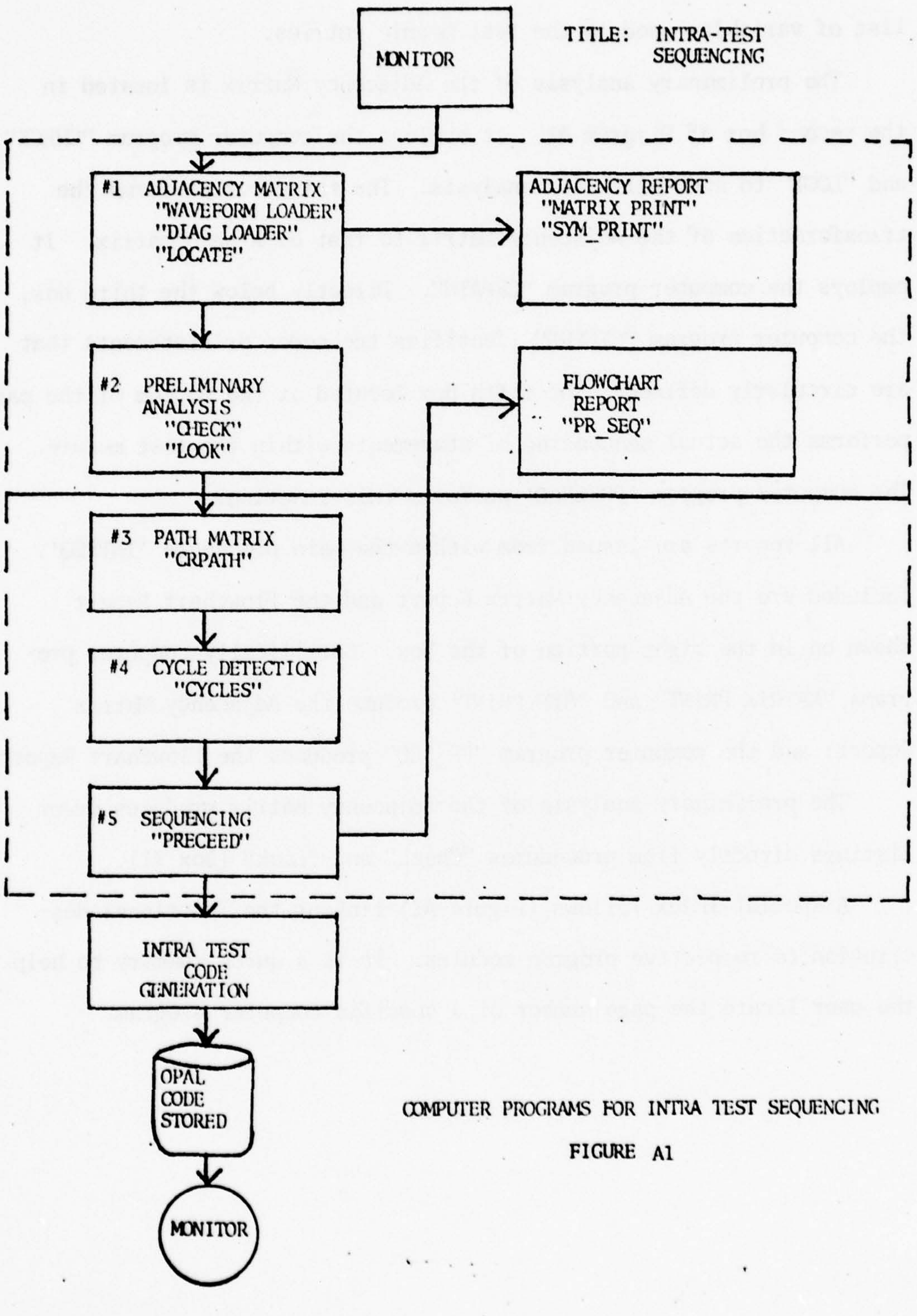
I would like to thank my advisor, Dr. Noah Prywes, for his leadership and direction throughout the duration of this research. Also, I wish to acknowledge Yung Chang for his collaboration in the design and implementation of this project.

APPENDIX A

A1 COMPUTER PROGRAMS FOR INTRA TEST SEQUENCING: INTRODUCTION

The diagram below (Figure A1) illustrates the organization of computer programs used to implement intra-test sequencing. The top portion of the diagram is the main procedure which issues the control logic to invoke specific modules performing unique sequencing functions. The bottom portion of the diagram presents three separate boxes each performing tasks used in this phase and shared by other phases of the NOPAL processor. Together the top and bottom portions constitute the five phases of intra-test sequencing. The objective of each box is written in the first line followed by the corresponding computer program names. Each program name is surrounded by double quotes to easily distinguish their presence in the diagram. This diagram arrangement is functionally similar to the center partition of the NOPAL Processor Diagram showing Intra Test Analysis (Figure 1.2). While both include functional performance at each phase, Diagram A1 is more comprehensive. Also stating those computer programs responsible for the implementation.

The box labelled "INTSEQ" is the main control procedure of the process. It contains sub-procedures that are exclusively used for intra test sequencing. Three procedures, "WAVEFORM LOADER","DIAG LOADER", and "LOCATE" exclusively construct the adjacency matrix. Together they are shown in the first box in Diagram A1, which corresponds to the first box of the center partition of Diagram 1.2. "WAVEFORM LOADER" retrieves conjunctions and assertions from the associative memory to collect a list of variables used. "DIAG LOADER" retrieves diagnoses from the associative memory, also collecting a list of variables used. Both of these programs rely upon "LOCATE" to eliminate redundant entries to produce a complete


MONITOR
TITLE: INTRA-TEST SEQUENCING
#1 ADJACENCY MATRIX "WAVEFORM LOADER" "DIAG LOADER" "LOCATE"
ADJACENCY REPORT "MATRIX PRINT" "SYM_PRINT"
#2 PRELIMINARY ANALYSIS "CHECK" "LOOK"
FLOWCHART REPORT "PR_SEQ"
#3 PATH MATRIX "CRPATH"
#4 CYCLE DETECTION "CYCLES"
#5 SEQUENCING "PRECEED"
INTRA TEST CODE GENERATION
OPAL CODE STORED
MONITOR

COMPUTER PROGRAMS FOR INTRA TEST SEQUENCING

FIGURE A1

list of variables used in the test module entries.

The preliminary analysis of the Adjacency Matrix is located in the second box of Diagram A1. It employs the computer program "CHECK" and "LOOK" to accomplish its analysis. The third box performs the transformation of the Adjacency Matrix to that of a path matrix. It employs the computer program "CRPATH". Directly below the third box, the computer program "CYCLES" identifies the nodes or statements that are circularly defined. The fifth box located at the bottom of the page performs the actual sequencing of statements within the test module. The computer program 'PRECEED' performs this function.

All reports are issued from within the main procedure 'INTSEQ'. Included are the Adjacency Matrix Report and the Flowchart Report shown on in the right portion of the box. Specifically computer programs "MATRIX PRINT" and 'SYM_PRINT' produce the Adjacency Matrix Report; and the computer program "PR_SEQ" produces the Flowchart Report.

The preliminary analysis of the adjacency matrix produces error listings directly from procedures "Check" and "Look" (Box #2).

A special index follows (Figure A2) linking the functional description to respective program modules. It is a quick summary to help the user locate the page number of a specific computer program.

INTRA TEST SEQUENCING

FUNCTION	COMPUTER MODULES	PAGE	COMMENTS
invoking intra test sequencing	INSEQ	A5	contains data structures and control logic for sequencing
construct the adjacency matrix	WAVEFORM LOADER DIAG LOADER LOCATE	A10 A11 A12	
preliminary analysis of the adjacency matrix	CHECK LOOK	A14 A16	
adjacency matrix report	MATRIX PRINT SYM PRINT	A17 A18	set SEQOPT = 1 when invoking NOPAL processor
construct path matrix	CRPATHS	A24	
locate cycles	CYCLES	A25	
sequencing	PRECEED	A27	
sequencing report	PR SEQ	A19	set SEQOPT = 1 or 2 when invoking NOPAL processor

FUNCTIONAL DESCRIPTION AND PROGRAM MODULES

FIGURE A2

```
INTSEQ:  PROC(BEGIN);
/*** UNICOLL VERSION ***/
/*THIS PROCEDURE CREATES THE ADJACENCY MATRIX USED FOR
INTRA MODULE  SEQUENCING.  THE ACTUAL MATRIX IS A CONTROLLED
VARIABLE STRUCTURE: THUS OPTIMIZING ON THE SYSTEM MEMORY.
IT BEGINS BY COUNTING THE NUMBER OF WAVEFORMS (BOTH STIMULI
AND MEASUREMENT),VARIABLES(BOTH SOURCE AND TARGET), AND
DIAGNOSES TO DETERMINE THE SIZE OF THE MATRIX.
ONCE THE ADJACENCY MATRIX IS CONSTRUCTED, ERROR DETECTION AND
INTERNAL SEQUENCING FOLLOW..

THERE ARE THREE NEW DATA STRUCTURES.  THEY ARE 'A','HOMER', AND
    A -A CONTROLLED STRUCTURE WHOSE SIZE IS DETERMINED BY SUM TOTAL
        OF THE NUMBER OF DIAG'S,VARIABLES, AND WAVEFORMS.
    HOMER- A CONTROLLED STRUCTURE THAT POINTS BACK TO THE
        DIRECTORY (KEYNODE).  THIS ENABLES EVERY ADDRESS IN A THE
        ABILITY TO TRACE BACK TO ITS STORAGE ENTRY IN THE  DIRECTORY.
    VAR-A CONTROLLED STRUCTURE USED TO COLLECT ALL SOURCE AND TARGET
        REFERENCES.  THE FIRST COLUMN INDICATES 'SOURCE';
        THE SECOND INDICATES 'TARGET'.  THE FIRST ROW
        OF BOTH COLUMNS STORES THE NUMBER OF VARIABLES PLACED
        IN EACH COLUMN.
DIAG-A CONTROLLED STRUCTURE USED TO COLLECT ALL DIAGNOSE
      REFERENCES. THE  FIRST ROW INDICATES THE NUMBER OF DIAG LBLS.
                                                  */

DCL 1 SYMBOL(       200 ) EXT,        /* 2-TABLE DIRECTOR */
        2 KEYNAME CHAR(        12 ) VAR,
        (2 UP,  2 DOWN,  2 TYPLIST)  FIXED BIN;
DCL 1 KEYNODE(       300 ) EXT,
        (2 KEYTYPE,  2 TYPLINK,  2 NEXTYPE,  2 HOME) FIXED BIN,
        2 REFLIST PTR;
DCL 1 STORAGE_ENTRY BASED (STO_PTR) ,
        2 DATA PTR,  2 #KEYS FIXED BIN,
        2 KEYENTRY (N REFER (#KEYS)),
          3 NAME FIXED BIN,  3 NEXT PTR;
DCL 1 ANY BASED(DP) ,          /* INCLUDE TABLE_ENTRY  */
        (2 TYPE,  2 STMT#) FIXED BIN,
        (2 ALIAS,  2 ENTRY_SEQ#) FIXED BIN,
        2 FTYPE CHAR(1);
DCL 1 TEST BASED(DP) ,
        (2 TYPE,  2 STMT#) FIXED BIN,
        (2 STIM,  2 MEAS,  2 LOG) PTR;
DCL 1 WAVEFORMS ALIGNED BASED(DP) ,
        (2 TYPE,  2 STMT#) FIXED BIN,
        (2 S_LIST,  2 T_LIST) PTR,
        (2 TEST_LBL,  2 BREF_LBL) FIXED BIN,
        2 TRIPLET,
          3 POINTER PTR,  3 FLAG BIT(1);
DCL 1 WAVEFORMS_LEVEL BASED(TP) ,
        2 #WAVEFORMS FIXED BIN,
        2 ENTRY(N REFER(#WAVEFORMS)),
          3 WAVEFORM PTR,  3 LEVEL FIXED BIN;
DCL 1 DIAGNOSIS BASED(DP) ,
        (2 TYPE,  2 STMT#) FIXED BIN,
          2 USED PTR,
          2 OP_MSG,
            3 AFFECTED_COMP PTR,  3 OTHER_PARMS PTR,
            3 TYPE FIXED BIN,
            3 TIMING,
              4 VALUE DEC FLOAT,  4 DIM CHAR(12),
          2 OP_RPS,
            3 VAR_PTR PTR,  3 Y_N CHAR(1);
      DCL 1 MSG_PARM BASED (TP) ,
          2 #PARMS FIXED BIN,
          2 PARM (NC REFER(MSG_PARM.#PARMS)),
            3 TYPE FIXED BIN, 3 PNTR  PTR;
      DCL 1 LOGIC BASED (DP),
          (2 TYPE, 2 STMT#) FIXED BIN,
          2 #ENTRY FIXED BIN,
          2 ENTRY (NC REFER(#ENTRY)),
          3 OPERATOR CHAR(4),
          3 DIAG_LBL FIXED BIN;
      DCL 1 STIM_MEAS BASED (DP),
       (2 TYPE, 2 STMT#) FIXED BIN,
        2 WAVEFORMS_PTR PTR;
      DCL 1 STR BASED (TP) ,
          2 #CHAR FIXED BIN,
          2 TEXT CHAR(LS REFER (STR.#CHAR));
      DCL PLACE FIXED BIN;
      DCL 1 LIST_VAL BASED(TP) ,
            2 #VAL FIXED BIN,
            2 VAL(N REFER(LIST_VAL.#VAL)) FIXED BIN;
      DCL 1 LIST_PTR BASED(TP) ,
            2 #PTR FIXED BIN,
            2 PTR(N REFER(LIST_PTR.#PTR)) PTR;
      DCL 1 DCL ALIGNED BASED(TP) ,
           2 ID FIXED BIN,  2 DESCRIPTOR PTR,
            2 SCOPE BIT(1),  2 LINK PTR;
      DCL 1 IF_CELL ALIGNED BASED (TP),
          (2 TRUE_PART, 2 FALSE_PART) PTR,
          2 CF BIT(1), 2 #CHAR FIXED BIN,
          2 CONDITION CHAR(LS REFER(IF_CELL.#CHAR));
      DCL 1 SIMPLE_CONJ BASED (TP),
           2 #TRIPLET FIXED BIN,
           2 TRIPLET (ND REFER(#TRIPLET)),
           3 CDE PTR, 3 RELATION CHAR(1), 3 FDE PTR;
      DCL $A FIXED BIN EXT;    /*  FOR ALPHAMERIC (=2) */
      DCL NAMES(31) CHAR(       12 ) VAR EXT;
      DCL TYPES(31) FIXED BIN EXT;
      DCL (STO_PTR,DP,TP,SP) PTR  ;
      DCL (SPEC#,TEST#,STIM#,MEAS#,DIAG#,MSG#,LOGIC#,CONJ#,ASRT#,
          COMP#,CMPFL#,OUTPT#,ATEPT#,FUNC#,VAR#,END#) FIXED BIN EXT;
      DCL RPTRS(       200 ) PTR;

      DCL  A(N,N) BIT(1) INIT((N*N)('0')) CONTROLLED ;
  DCL P (N,N) BIT(1) CONTROLLED;
  DCL ORDER(N) FIXED BIN EXT CONTROLLED;
      DCL $SYM(N) CHAR(       12 ) VARYING CONTROLLED;
      DCL HOM_NODE(SIZE) FIXED BIN CONTROLLED;
      DCL HOM_PTR(SIZE) PTR CONTROLLED;
      DCL VNODE(200) FIXED BIN; DCL VPTR(200) PTR;
      DCL (ERR_CNT,NO_WARN) FIXED BIN EXT;
```

```
DCL (#V,#MEAS,#DIAG,#STIM,#W,N,SIZE,HOM) FIXED BIN;
DCL SEQOPT FIXED BIN EXT;
DCL DIAG(       200 ) PTR;
DCL (MEAS_CONJ,STIM_CONJ) FIXED BIN INIT(0);
DCL(BEGIN,T_PTR,TP_LEVEL,W_PTR,P3_PTR) PTR;
DCL (TTP,TPM_LEVEL,TPS_LEVEL,NEW_VAR) PTR;
DCL (TP_PARMS,TP_OPERATOR) PTR;

DCL (         $NAME,$CONJ,$ASRT,$DIAG) CHAR(          12 ) VARYING;
DCL $TYPE CHAR(15) VARYING,$I CHAR(4) VARYING,
$ORDER CHAR(4) VARYING, $RANK CHAR(4) VARYING;
DCL ($MAT,$VAR,$TST) CHAR(12) VARYING;
DCL (LIN1,LIN2,LIN3) CHAR(90) VARYING;
DCL TEMP PTR;  DCL $V1 CHAR(60) VARYING;
DCL TEN FIXED BIN;
DCL $C PICTURE '9';
DCL $T PICTURE '9';
DCL $U CHAR(1) INIT('-');
DCL BLK1 CHAR(1) INIT(' ');
DCL SEQRPT FILE OUTPUT PRINT ENV(F(131));
DCL SEQERR  FILE OUTPUT PRINT ENV(F(131));
OPEN FILE (SEQRPT), FILE (SEQERR);
$ASRT = 'ASSERTION';   $VAR = 'VARIABLE';
$DIAG = 'DIAGNOSES';   $MAT ='SUBSCRPT_VAR';
$CONJ = 'CONJUNCTION';


/** CALCULATE # OF WVEFORMS AND STORE VARIABLES **/
STO_PTR = BEGIN;  DP,T_PTR = DATA;
$TST = KEYNAME(HOME(NAME(1)));
#STIM,#MEAS = 0;

CALL WAVEFORM_LOADER ('1'B);

/*** LOAD DIAG WITH STORAGE POINTERS TO DIAGNOSES ***/
STO_PTR=BEGIN;  DP=DATA;
STO_PTR=LOG;  DP=DATA;
IF STO_PTR = NULL THEN #DIAG=0;
ELSE #DIAG=LOGIC.#ENTRY;

DO I = 1 TO #DIAG;
TYPES(1)=NAME(DIAG_LBL(I));
CALL RETREVS('0'B,RPTRS,NR,DIAG#);
DIAG(I)=RPTRS(1);
END;

/** FOR EACH DIAG, SAVE OPERATOR AND OTHER PARMS **/
CALL DIAG_LOADER('1'B);

SIZE,N = #DIAG + #V + #STIM + #MEAS;
ALLOCATE HOM_NODE,HOM_PTR,A;
HOM_NODE = 0;
COUNT   = #STIM + #MEAS + #DIAG; A='0'B;
```

```
/** MOVE TEMP VARIABLES TO HOMER **/
DO I=1 TO #V;
COUNT = COUNT +1;
HOM_NODE(COUNT) = VNODE(I);
HOM_PTR(COUNT) = VPTR(I);
END;
HOM= #STIM + #MEAS + #DIAG;
COUNT,STIM_CONJ,MEAS_CONJ = 0;

/** LOAD HOMER WITH WAVEFORMS, AND 'A' WITH VARIABLES **/
CALL WAVEFORM_LOADER('0'B);

/**LOAD HOMER WITH DIAG **/
CALL DIAG_LOADER('0'B);
#W = #STIM + #MEAS;
ALLOCATE $SYM;

IF SEQOPT  = 1 THEN CALL MATRIX_PRINT;
   ELSE CALL SYM_PRINT;
CALL CHECK;


 /** DETECT ERRORS AND SEQUENCE  **/
/****** CALL INTERNAL CODE GENERATION BEFORE RETURNING *************/

 /** CREATE PATH MATRIX  **/
 ALLOCATE P;  CALL CRPATHS (A,P,N);

 /** CHECK DIAGONAL FOR CYCLES  **/
 DO I=1 TO N;
 IF P(I,I) = '1'B THEN
    DO; CALL CYCLES(A,P,N,$SYM);
    RETURN;
    END;

 END;

 /** PRECEDENCE **/
 ALLOCATE ORDER;  CALL PRECEED(A,ORDER,N);
 IF SEQOPT <=2 THEN CALL PR_SEQ(ORDER,$SYM,N);
 FREE A,HOM_NODE,HOM_PTR,$SYM,P;
```

```
WAVEFORM_LOADER: PROC(SWITCH);
  /** THIS PROCEDURE WILL EXAMINE ALL WAVEFORMS.
  IF 'SWITCH' IS SET TO '0', ALL NEW SOURCE AND
  TARGET VARIBLES ARE INSERTED TO THE TEMPORARY
  VARIABLE LIST. IF SWITCH IS SET TO '1' THE WAVEFORM
  STORAGE POINTER IS LOADED TO 'HOMER', AND ALL
  LOCATIONS IN THE ADJACENCY MATRIX ARE SET...*/


  DCL SWITCH BIT(1);
  COUNT = 0;
  DP = T_PTR;
  DO I=1 TO 2;
  IF I=1 THEN STO_PTR = T_PTR->TEST.STIM;
  ELSE STO_PTR = T_PTR->TEST.MEAS;
  IF STO_PTR=NULL THEN GOTO BAD;
  DP = DATA;  /* FOLLOW TO STIM AND MEAS */

  TP = WAVEFORMS_PTR;
  IF I=1 THEN #STIM = #WAVEFORMS;
  ELSE #MEAS = #WAVEFORMS;
  TTP = TP;

     DO J=1 TO #WAVEFORMS;
     STO_PTR = TTP->WAVEFORMS_LEVEL.WAVEFORM(J);
     W_PTR,DP = DATA;
     IF ¬SWITCH THEN /**STIM OR MEAS CONJ NOT NULL **/

        DO; COUNT = COUNT +1;
        HOM_PTR(COUNT)=STO_PTR;
        HOM_NODE(COUNT)=0;
        IF ANY.TYPE ¬= CONJ# THEN GOTO BY;
        IF J=1 & I=1 THEN STIM_CONJ=COUNT;
        IF J=1 & I=2 THEN MEAS_CONJ=COUNT;
        END;
   BY:
        IF J=2 & I=2 THEN IF STIM_CONJ >0 &
        MEAS_CONJ >0 THEN A(STIM_CONJ,MEAS_CONJ) = '1'B;

        DO K = 1,2 ;
        IF K=1 THEN TP=W_PTR->WAVEFORMS.S_LIST;
        ELSE TP = W_PTR->WAVEFORMS.T_LIST;

           DO WHILE (TP¬=NULL);
           NODE = NAME(DCL.ID);
           PLACE = LOCATE(NODE,TP,SWITCH);
           IF ¬SWITCH THEN IF K=1 THEN A(PLACE,COUNT)='1'B;
           IF ¬SWITCH THEN IF K=2 THEN A(COUNT,PLACE)='1'B;
           TP=LINK;
           END;

        END;

     END;
BAD:

  END;
  END WAVEFORM_LOADER;
```

```
DIAG_LOADER: PROC(SWITCH);
   /** THIS PROCEDURE WILL EXAMINE ALL DIAGNOSES.
   IF SWITCH IS SET TO '0', ALL NEW DIAGNOSES' VARIABLES
   ARE INSERTED TO THE TEMPORARY VARIABLE LIST. IF
   SWITCH IS SET TO '1', THE DIAGNOSES STORAGE POINTER
   IS LOADED TO HOMER, AND ALL LOCATIONS IN THE ADJACENCY
   MATRIX ARE SET......          */


   DCL SWITCH BIT(1);
   COUNT  = #STIM + #MEAS;
   DO I=1 TO #DIAG;
   STO_PTR = DIAG(I); DP = DATA;

   IF ¬SWITCH THEN
      DO; COUNT = COUNT + 1;
      HOM_PTR(COUNT) = STO_PTR;

  HOM_NODE(COUNT) = 0;
  END;

  DO J = 1,2 ;
  IF J=1 THEN TP=OTHER_PARMS;
  ELSE TP = OP_RPS.VAR_PTR;

  IF J=1 THEN    /** OTHER PARMS **/
     DO K=1 TO #PARMS WHILE (TP¬=NULL);
     IF MSG_PARM.TYPE(K)=$A THEN
        DO;
        SP = MSG_PARM.PNTR(K);
        NODE = NAME (SP->DCL.ID);
        PLACE = LOCATE (NODE,SP,SWITCH);
        IF ¬SWITCH THEN A(PLACE,COUNT) = '1'B;
        END;

     END;

  IF J=2 THEN    /** OPERATOR RESPONSE **/
     DO WHILE (TP¬=NULL);
     NODE = NAME(DCL.ID);
     PLACE = LOCATE (NODE,TP,SWITCH);
        IF ¬SWITCH THEN A(COUNT,PLACE) = '1'B;
     TP = LINK;
     END;

  END;
END;
END DIAG_LOADER;
```

```
LOCATE: PROCEDURE (NEW_VAR,NEW,OPTION) RETURNS (FIXED BIN);
  /**********************************************************
  THIS PROCEDURE WILL LOCATE THE POSITION OF A VARIABLE IN
  EITHER 'HOMER' OR 'VAR'. 'VAR' IS USED TO DERTMINE THE ACTUAL
  NUMBER OF UNIQUE VARIABLES.. ONCE THIS HAS BEEN DETERMINED
  'HOMER' IS  DYNAMICALLY ALLOCATED WITH THAT EXACT  NUMBER.
  THUS THE PROCEDURE HAS TWO ABILITIES: ONE TO SEARCH THE
  TEMPORARY VARIABLE LIST, AND ADD NEW VARIABLES AS THEY
  COME, AND(2) TO SEARCH THE FINAL VARIABLE LIST 'HOMER'
  AND RETURN ITS POSITON SO THAT IT MIGHT  BE ADDED TO THE
  ADJACENCY MATRIX. THE FOLLOWING CODE ALTHOUGH SHORT WAS
  DEVELOPED INTO ITS OWN PROCEDURE BECAUSE ITS BASICLY
  INIATES A SEARCH CHECKING FOR BOTH SIMPLE AND COMPLEX
  VARIABLES.

  INPUT PARAMETERS:
     NEW_VAR - STO_PTR TO VARIABLE NAME
     TP      - TP POINTER TO DCL
     OPTION  - IF '0' WILL SEARCH 'HOMER' AND RETURN POSITON
               IF '1' WILL SEARCH 'VAR' AND INSERT IF NEW
  *********************************************************/

  DCL (TP_OLD,TP_NEW,OLD_SP,NEW_SP,NEW,OLD) PTR;
  DCL (K,NEW_VAR,TEMP,MAX) FIXED BIN;
  DCL  OPTION BIT(1);  MAX=0;

  DO L = 1 TO #V;
  MAX = L;
  IF OPTION THEN TEMP = VNODE(L);
  ELSE TEMP = HOM_NODE(HOM + L);
  IF NEW_VAR ¬= TEMP THEN GOTO END_L;
  IF OPTION THEN OLD = VPTR(L);
  ELSE OLD = HOM_PTR(HOM + L);

       /** CHECK FOR SIMPLE VARIABLE **/
   IF NEW->DCL.DESCRIPTOR = NULL &
       OLD->DCL.DESCRIPTOR = NULL
       THEN GOTO MATCH;
          /** CHECK THAT BOTH ARE ARRAYS **/
   IF NEW->DCL.DESCRIPTOR ¬= NULL &
       OLD->DCL.DESCRIPTOR ¬= NULL THEN

       DO;  /*** CHECK THAT SUBSCRIPTS MATCH ***/
          TP_OLD = OLD->DCL.DESCRIPTOR;
          TP_NEW = NEW->DCL.DESCRIPTOR;
          IF TP_NEW->LIST_PTR.#PTR ¬=
              TP_OLD->LIST_PTR.#PTR THEN GOTO INSERT;
           DO M = 1 TO TP_NEW->LIST_PTR.#PTR;
            OLD_SP = TP_OLD->LIST_PTR.PTR(M);
            NEW_SP = TP_NEW->LIST_PTR.PTR(M);
            IF OLD_SP->STR.TEXT ¬= NEW_SP->STR.TEXT
               THEN GOTO INSERT;
            END;  /** END M = 1 TO # PTR **/
         GOTO MATCH;
      END;          /** END BOTH ARRAYS *****/
      ELSE GOTO INSERT;

 END_L:  END;
```

```
INSERT: IF OPTION THEN /* INSERT */
  DO;  #V = #V + 1; VNODE(#V) = NEW_VAR;
  VPTR(#V) = NEW;  RETURN(#V);
  END;
  ELSE CALL SYSERR ('SEARCHED VARIABLE NOT IN HOMER'); /
MATCH:  MAX=MAX + HOM;  RETURN (MAX);
END LOCATE;

CHECK: PROC;
  /*****************************************************
  THIS PROCEDURE WILL EXAMINE  THE ADJACENCY MATRIX FOR:
    1)  MORE THEN 1 TARGET PER ASSERTION
    2)  MULTIPLE DEFINED TARGETS
    3)  INVALID LEFT HAND EXPRESSIONS
    4)  INVALID OPERATORS
  *****************************************************/

  /*** MULTIPLE DECLARATIONS OF TARGET ***/
  HOM1 = HOM +1;

  DO I=HOM1 TO N;
  LIN1 ='';  #E=0;

     DO J=1 TO HOM;
     IF A(J,I) = '1'B THEN

        DO; #E=#E +1;
        /** STORE WAVEFORMS **/
        LIN1 = LIN1 || $SYM(J) || ' ,' ;
        END;

     END;

  IF #E >1 THEN
     DO:  IZ=LENGTH(LIN1); LIN1=SUBSTR(LIN1,1,IZ-1);
        PUT FILE (SEQERR) EDIT
     ('*WARNING (POSSIBLE AMBIGUITY): VARIABLE ' || $SYM(I)
     ||' OF TEST ' || $TST ||' ,'
     ||' IS  DEFINED MORE THAN ONCE IN ' ||  LIN1
     ||'.  THEY MUST BE EXCLUSIVE .') (SKIP,A);
     NO_WARN = NO_WARN + 1;
     END;

  END;
  /*** MORE THEN 1 TARGET IN ASSERTION ***/

  DO I=1 TO #W;
  LIN1='';
  STO_PTR = HOM_PTR(I); DP=DATA; #E=0;
  IF WAVEFORMS.TYPE ¬= ASRT# THEN GOTO NO_ASRT#;

  DO J=HOM1 TO N;
     IF A(I,J) = '1'B THEN

        DO; #E=#E +1;
        LIN1 = LIN1 || $SYM(J) || ' ,' ;
        END;

     END;
  IF #E >1 THEN
     DO:  IZ=LENGTH(LIN1): LIN1=SUBSTR(LIN1,1,IZ-1);
        PUT FILE (SEQERR) EDIT
     ('*ERROR (AMBIGUITY): IN ASSERTION ' || $SYM(I)
     ||' OF TEST ' || $TST ||', THERE ARE  TWO OR MORE'
     ||' TARGET VARIABLES: ' ||LIN1) (SKIP,A);
     ERR_CNT = ERR_CNT + 1;
     END;

  /*** EXAMINE LEFT EXPRESSION AND OPERATOR ***/
  SP=POINTER; IF #E >0 THEN

     DO: IF TRIPLET.FLAG = '0'B THEN CALL LOOK(SP);
        ELSE DO:
        COND: CALL LOOK(SP->IF_CELL.TRUE_PART);
        IF SP->IF_CELL.CF='1'B THEN
           DO: SP=SP->IF_CELL.FALSE_PART; GOTO COND;
           END;
           ELSE IF SP->IF_CELL.FALSE_PART ¬= NULL THEN
           CALL LOOK (SP->IF_CELL.FALSE_PART);

        END;

     END;

NO_ASRT#:  END;

     LOOK: PROCEDURE(TEMP);
      DCL (TEMP,SPP) PTR;
      DCL 1 SIMPLE_ASRT BASED (TEMP),
          2 RELATIONAL,
             3 EXPR(2) PTR, 3 OPERATOR CHAR(2),
          2 RANGE,
             3 EXP PTR, 3 PCNT CHAR(1);

      SPP = EXPR(1): LIN1='';  #E=0;
      DO J=HOM1 TO N;
      IF A(1,J)='1'B THEN
         DO: IF SPP->STR.TEXT=$SYM(J) THEN GOTO OPER;
         #E=#E +1;
         IF #E = 1 THEN LIN1=$SYM(J);
         END;
      END;

      PUT FILE(SEQERR) EDIT
      ('*ERROR (AMBIGUITY): EXPRESSION ' || SPP->STR.TEXT
      ||' PRECEEDING THE ''='' IN ASSERTION ' || $SYM(I)
      ||' OF TEST ' || $TST || ' DOES NOT MATCH THE TARGET VARIABLE '
      || LIN1) (SKIP,A);
      ERR_CNT = ERR_CNT + 1;

    OPER: IF SIMPLE_ASRT.OPERATOR ¬= '=' THEN
         DO: SIMPLE_ASRT.OPERATOR = '=';
         PUT FILE (SEQERR) EDIT
         ('*WARNING (INCONSISTENCY): IN ASSERTION ' || $SYM(I)
         ||' OF TEST ' || $TST ||', A VARIABLE IS DECLARED AS'
         ||' TARGET; BUT THE RELATION OPERATOR IS  NOT AN EQUAL SIGN.'
         ||'  REPLACED BY AN EQUAL SIGN.') (SKIP,A);
         NO_WARN = NO_WARN + 1;
         END;

      END LOOK;
      END CHECK;
```

```
MATRIX_PRINT: PROC;

  STO_PTR = BEGIN;
  PUT FILE(SEQRPT) PAGE;
  PUT FILE(SEQRPT)    EDIT ('INTRA MODULE SEQUENCING ' || $TST,
  'ANALYSIS OF THE ADJACENCY MATRIX')
  (SKIP(5),COL(26),A,SKIP(1),COL(26),A,SKIP(3));
  IF N<1 THEN RETURN;
  LIN1,LIN2,LIN3 = ' ';

  IF N>94 THEN
     DO;  /**TO BIG TO PRINT **/
     PUT FILE (SEQRPT) EDIT ('ERROR: ATTEMPT TO WRITE MATRIX '
     ||'WHOSE SIZE EXCEEDS MAXIMUM # OF 94. THE SAME INFORMATION '
     ||'IS AVAILABLE IN THE INTERNAL SEQUENCING REPORT WHICH DOES'
     ||' NOT HAVE A SIZE LIMIT') (SKIP,A,SKIP);
     RETURN;
     END;


  IF N<47 THEN B=2; ELSE B=1;
  COUNT=0;   TEN = 0;

  DO I=1 TO N;
  COUNT = COUNT  + 1;
  IF COUNT = 10 THEN
     DO; COUNT = 0;   TEN = TEN + 1;
     END;
  $C = COUNT;   $T = TEN;

  IF I<10 THEN LIN1 = LIN1 || BLK1;
  ELSE LIN1 = LIN1 || $T;
  LIN2 = LIN2 || $C;
  LIN3 = LIN3 || $U;
  IF N<47 THEN
     DO; LIN1 = LIN1 || BLK1;
     LIN2 = LIN2 || BLK1;
     LIN3 = LIN3 || BLK1;
     END;
  END;
  IF N>9 THEN PUT FILE (SEQRPT)    EDIT (LIN1) (COL(35),A,SKIP);
  PUT FILE (SEQRPT)    EDIT (LIN2) (COL(35),A,SKIP);
  PUT FILE (SEQRPT)    EDIT (LIN3) (COL(35),A,SKIP);
  CALL SYM_PRINT;
  END MATRIX_PRINT;
```

```
SYM_PRINT:  PROCEDURE;
 /*** PRINT NAMES AND SAVE THEM FOR 'CHECK' PROCEDURE ***/

 DO I = 1 TO N;
 IF HOM_NODE(I) > 0 THEN
    DO; $NAME = KEYNAME(HOME(HOM_NODE(I)));
    $TYPE = $VAR;  TP = HOM_PTR (I);
    IF DESCRIPTOR = NULL THEN GOTO PNT;
```

```
      /** SUBSCRIPTED VARIABLE  **/
      TEMP = DESCRIPTOR;  $V1 = $NAME;
         DO J = 1 TO TEMP->LIST_PTR.#PTR;
         SP = TEMP->LIST_PTR.PTR(J);
         IF J=1 THEN $V1 = $V1 || '(' || SP->STR.TEXT;  ELSE
         $V1 = $V1 || ' ,' || SP->STR.TEXT;
         END;

      $NAME = $V1 || ')' ;
      GOTO PNT;
      END;

   STO_PTR = HOM_PTR(I); DP=DATA; $TYPE = 'ERROR';

   IF ANY.TYPE = ASRT# THEN $TYPE = $ASRT;
   IF ANY.TYPE = CONJ# THEN $TYPE = $CONJ;
   IF ANY.TYPE = DIAG# THEN $TYPE = $DIAG;

   $NAME = KEYNAME(HOME(NAME(1)));
PNT: IF SEQOPT  = 1 THEN
   PUT FILE(SEQRPT)   EDIT (I,$NAME,$TYPE,(A(I,J)DO J=1 TO N))
   (COL(2),F(4),COL(8),A,COL(22),A,COL(36),(N)A(B));
   $SYM(I) = $NAME;
   END;
   END SYM_PRINT;


PR_SEQ:  PROC (ORDER,DICT,N);
DCL CAT ENTRY  (CHAR(*),FIXED BIN, FIXED BIN) RED,
CON ENTRY(FIXED BIN,FIXED BIN)RED ;
/** PROCEDURE TO PRINT SEQUENCING   **/
DCL (IORDER,IRANK) FIXED BIN INIT(0);
   DCL #OUT FIXED BIN INIT(0);
DCL KEYWD CHAR(5) VARYING;
DCL FLOATMX FLOAT INIT(1E+75) STATIC,
    LTEMP FIXED BIN, TEMP CHAR(70 ) VAR INIT(''),
    CURTXT CHAR(140) VAR INIT(''),
    SEMICOLON CHAR(1) INIT(';'),
    LPAR  CHAR(1) INIT('('), RPAR CHAR(1) INIT(')'),
    COLON CHAR(1) INIT(':'), COMMA CHAR(2)  INIT(', '),
    BLANKS CHAR(70) INIT((70)' '), TAG BIT(1),
    ORDER(*) FIXED BIN, DICT(*) CHAR(12) VARYING,
    RANK(N) FIXED BIN EXT CONTROLLED, PARCMA CHAR(3) INIT('), '),
   B1 CHAR(1) INIT(' '), SKP FIXED INIT(1);

DCL 1 SIMPLE_ASRT BASED (TP),
   2 RELATIONAL,
     3 EXPR(2) PTR, 3 OPERATOR CHAR(2),
   2 RANGE,
     3 EXP PTR, 3 PCNT CHAR(1);

DCL 1 CONNECTORS BASED (TP),
   2 DIM CHAR(12),
   2 #PTS FIXED BIN,
   2 POINTS (NP REFER(CONNECTORS.#PTS)) FIXED BIN;

DCL 1 AFFECTED_COMP BASED(TP),
    2 AND_OR CHAR(1),2 #COMPS FIXED BIN,
    2 ELEMENT(NC REFER (#COMPS)),
      (3 COMP_FL#, 3 COMP_ID, 3 FAIL_FN) FIXED BIN;
```

```
IF SEQOPT =  2 THEN PUT FILE(SEQRPT)  PAGE;
PUT FILE (SEQRPT) EDIT('SEQUENCE OF PROCESSING ' ||
'FOR TEST '|| $TST) (SKIP(3),COL(26),A,SKIP(2));
PUT FILE (SEQRPT) EDIT('ORDER','VECT','ORDER','RANK','NAME',
 'TYPE','TEXT','INDEX','VECTOR')
(COL(1),A,SKIP(1),COL(1),A,COL(7),A,COL(15),A,COL(27),
A,COL(44),A,COL(70),A,SKIP(1),COL(1),A,COL(7),A,SKIP(2));

DO M=1 TO N;
I=ORDER(M);  /** I IS THE MATRIX NODE BEING EXAMINED **/
IF HOM_NODE(I) >0 THEN CALL VAR_PT; ELSE
   DO;   /** WAVEFORMS OR DIAGNOSES ***/
     STO_PTR=HOM_PTR(I);  OP=DATA;  $TYPE='ERROR';
     IORDER = I;  IRANK = RANK(I);
     $NAME = DICT(I);  HOM1= #W + #DIAG +1;
     IF ANY.TYPE = DIAG# THEN CALL DIAG_PT; ELSE CALL WFORM_PT;
     IF LTEMP >0 | (LENGTH($NAME)) >0 THEN CALL CON(0,2);
   END;   /** END WAVEFORM OR DIAGNOSES  ***/
   CALL CON(0,2) ;
END;   /** END M ITERATION  ***/

 VAR_PT: PROC;

   TP=HOM_PTR(I);  S,T=0; LINE='';
   IF DESCRIPTOR=NULL THEN $TYPE=$VAR;
     ELSE $TYPE=$MAT;

   IF ¬SCOPE THEN TEMP='LOCAL';
   ELSE
   DO;  /** GLOBAL **/
      DO K=1 TO #W;
         IF A(K,I) THEN T=T+1;
      END;

   TEMP='GLOBAL /';
   IF T >0 THEN TEMP=TEMP ||' TARGET /';
   ELSE TEMP=TEMP || ' SOURCE /';
   END;
   PUT FILE (SEQRPT) EDIT(M,I,RANK(I),DICT(I),$TYPE,TEMP)
   (SKIP,COL(1),F(4),COL(7),F(4),COL(14),F(4),
   COL(26),A,COL(41),A,COL(58),A);
   $TYPE,TEMP='';
 END VAR_PT;

DIAG_PT: PROC;
    SKP,NB,K=1;  $TYPE=$DIAG;

    TP = OP_MSG.AFFECTED_COMP;
    IF TP¬=NULL THEN       /* AFFECTED COMPONENTS */
      DO; CALL CAT('AFFECTED COMPONENTS =',0,0);
         KEYWD = AND_OR || B1;
         DO L=1 TO #COMPS;
            KF = FAIL_FN(L);    CURTXT = KEYNAME(HOME(COMP_ID(L)));
            IF KF>0B THEN CURTXT = KEYNAME(HOME(KF))||LPAR||CURTXT
                                 || RPAR;
            CURTXT = CURTXT || KEYWD;      CALL CON(1, 0);
         END;
         SUBSTR(TEMP, LTEMP - 1) = COMMA;
      END;
      TP = OTHER_PARMS;
       IF TP¬=NULL THEN      /* OTHER PARAMETERS */
        DO: CALL CAT('OTHER PARAMETERS=(', 0 , 0);
            DO L=1 TO MSG_PARM.#PARMS;
            SP=MSG_PARM.PNTR(L);
             IF MSG_PARM.TYPE(L)¬=$A THEN
             CALL CAT(SP->STR.TEXT || COMMA,0,0);
            END;
            DO L=#W TO N;
            CURTXT=DICT(L) || COMMA;
            IF A(L,I) THEN CALL CON(0,0);
            END;
            TEMP = SUBSTR(TEMP, 1, LTEMP-2)  || PARCMA;
        END;

      IF OP_MSG.TYPE>0 THEN                /* OP. MSG. TYPE  */
      DO; CURTXT='TYPE = '||KEYNAME(HOME(NAME(OP_MSG.TYPE))) || COMMA;
          CALL CON(1,0);
      END;

      IF TIMING.VALUE<FLOATMX THEN       /* TIMING   */



           CALL CAT('TIME=' ||TIMING.VALUE || COMMA,0,0);

        TP = OP_RPS.VAR_PTR;
        IF TP¬=NULL THEN        /* OP. RESP. VARS */
          DO: CALL CAT('RESPONSE=(', 1 , 0);
                 DO L= HOM1 TO N;
                 IF A(I,L) THEN
                    DO; CURTXT=DICT(L) || COMMA; CALL CON(1,0); END;
                 END;
              TEMP = SUBSTR(TEMP,1,LTEMP-2) || PARCMA;
          END;

        IF OP_RPS.Y_N¬=B1 THEN              /*  Y/N  */
         CALL CAT('RESPONSE = ' || OP_RPS.Y_N || COMMA,1 ,0);
        LTEMP=LENGTH(TEMP);
        TEMP=SUBSTR(TEMP,1,LTEMP-2) || SEMICOLON;
END DIAG_PT;

WFORM_PT: PROC;
      TP = WAVEFORMS.POINTER;    TAG =¬FLAG;   STYPE = WAVEFORMS.TYPE;
      SKP,NB,KF=1; K=1B;
IF STYPE=CONJ# THEN $TYPE=$CONJ; ELSE $TYPE=$ASRT;
IF ¬TAG THEN CALL SIMPLE(TP,  NB);      /*SIMPLE CONJ/ASRT*/
ELSE  /* CONDITIONAL CONJ/ASRT */
  DO; COND:  CALL CAT('IF '||CONDITION|| ' THEN ',NB,0);
        TAG = IF_CELL.CF;
        CALL SIMPLE(TRUE_PART,  NB+4);     TP = FALSE_PART;

       IF TAG THEN
        DO; CALL CAT('ELSE ',NB,1);
              IF NB>KF THEN DO; NB = SKP;   K = 1;  END;
              ELSE DO; NB = NB + 5;   K = 0;  END;
              GOTO COND;
         END;
        IF  TP¬=NULL THEN   /* ESLE=> SIMPLE CONJ. */
          DO: CALL   CAT('ELSE ',  NB,  1);
                CALL  SIMPLE(TP, NB + 4);
          END;
  END;    CURTXT='TARGET: ';
```

```
IF T_LIST ¬=NULL THEN
   DO:    CALL CAT('TARGET: ',SKP,1);
          DO L=HOM1 TO N;
          IF A(I,L) THEN
             DO; CURTXT=DICT(L) || COMMA;  CALL CON(1,0); END;
          END;
   SUBSTR(TEMP,LTEMP-1,1)=B1;
   END;

IF S_LIST ¬= NULL THEN
   DO:    CALL CAT('SOURCE: ',SKP,1);
          DO L= HOM1 TO N;
          IF A(L,I) THEN
             DO; CURTXT=DICT(L) || COMMA;  CALL CON(1,0); END;
          END;
   END;
  SUBSTR(TEMP, LTEMP-1, 1) = SEMICOLON;


  SIMPLE: PROC(P,NB);  /** OUTPUT SIMPLE CONJ/ASRT **/
   DCL (P, TP) PTR, NB FIXED BIN;
     IF STYPE=CONJ# THEN        /* TRIPLET CONJUNCTION */
       DO;
          DO L=1 TO  P->#TRIPLET;
           TP = P->CDE(L);     CALL CAT ('(<', NB, 0);
           DO J=1 TO TP->#PTS;
               CURTXT = KEYNAME(HOME(NAME(TP->POINTS(J)))) || COMMA;
                 CALL CON(NB, 0);
           END;
           SUBSTR(TEMP, LTEMP-1, 1) = '>';
           TEMP = TEMP || ' = ';
           TP = P->FDE(L);
           CALL CAT(TP->STR.TEXT || ') &', NB, 0);
          END;
          SUBSTR(TEMP, LTEMP, 1) = B1;
       END;
     ELSE       /*  ASRT   */
       DO; TP = P->EXPR(1);      CALL CAT(TP->STR.TEXT, NB, 0);
           TP = P->EXPR(2);
           CURTXT=B1||P->RELATIONAL.OPERATOR||B1||TP->STR.TEXT;
           CALL CON(NB,0);
           TP = P->RANGE.EXP;
           IF TP¬=NULL THEN
              DO; CURTXT=' +- ' || TP->STR.TEXT||P->PCNT;
              CALL CON(NB,0);  END;
       END;
   END SIMPLE;
  END WFORM_PT;
```

```
CAT:  PROC(TEXT, NB, OPT);    /* CONCATENATE & OUTPUT TEXT */
/** IF OPT = 0 CHECK TOTAL LENGTH, THEN CAT.
    IF OPT = 1 OUTPUT LAST RECORD, THEN CAT.
    IF OPT = 2 INITIAL                          */

DCL TEXT CHAR(*),(NB,OPT) FIXED BIN;
CURTXT = TEXT;

CON:  ENTRY(NB, OPT);    /* SAME AS CAT, EXCEPT TEXT FROM "CURTXT" */

    IF OPT=0 THEN
       DO;IF(LENGTH(TEMP))=0 THEN NB=0;
       IF(LENGTH(TEMP))+(LENGTH(CURTXT))<70 THEN
       TEMP=TEMP || SUBSTR(BLANKS,1,NB) || CURTXT ;
       ELSE OPT=1;
    END;

    IF OPT =1 | OPT=2 THEN
    DO; IF IORDER >0 THEN
      DO;   /*** PRINT ENTIRE LINE **/
      PUT FILE(SEQRPT) EDIT(M,IORDER,IRANK,$NAME,$TYPE,TEMP)
      (SKIP,COL(1),F(4),COL(7),F(4),COL(14),F(4),COL(26),A,
      COL(41),A,COL(58),A);
      $NAME,$TYPE='';
      IORDER,IRANK=0;
     END;  ELSE
     PUT FILE (SEQRPT) EDIT (TEMP) (SKIP,COL(58),A);
    IF OPT =2 THEN TEMP=''; ELSE TEMP=CURTXT;

     END;
       LTEMP=LENGTH(TEMP);
    END CAT;

  END PR_SEQ;
       END INTSEQ;
```

```
    CRPATHS: PROC(ADJMAT,PATHMAT,N);
          /*** DEBUG ***/
        DCL (ADJMAT,PATHMAT)(*,*) BIT(*);
         PATHMAT=ADJMAT;
    DO J=1 TO N;
        DO I=1 TO N;
              IF PATHMAT(I,J) THEN PATHMAT(I,*)=PATHMAT(I,*)|PATHMAT(J,*);
        END;
    END;
       END CRPATHS;
```

```
CYCLES: PROC (A,P,N,DICT);

        DCL SEQERR  FILE OUTPUT PRINT ENV(F(131));
         DCL (A,P) (*,*) BIT(*);
        DCL PRINT_LINE CHAR(125) VARYING;
        DCL USED (N) BIT(1);
       DCL DICT(*) CHAR(        12 ) VARYING;
      DCL (ROOT,REACHJ(N),PATH(N+1)) FIXED BIN;
      DCL PATH_EXTENDED BIT(1);

      DO ROOT=1 TO N;
         DO K=ROOT TO N;
           REACHJ(K)=ROOT;
           USED(K)='0'B;
         END;

         LEVEL=1;
         PATH(1)=ROOT;
         I=ROOT;

         DO WHILE(LEVEL¬=0);
         PATH_EXTENDED='0'B;
         JMIN=REACHJ(I);
         DO J=JMIN TO N WHILE( (LEVEL¬=0)&¬PATH_EXTENDED);
            IF A(I,J)&P(J,ROOT)&¬USED(J) THEN DO;
               PATH_EXTENDED='1'B;
               CALL EXTEND_PATH;
               IF J=ROOT THEN DO;
                  CALL PRCYCLE(PATH,LEVEL);
                  CALL BACKTRACK;
                  END;
               END;
        END;
         IF ¬PATH_EXTENDED THEN DO;
            REACHJ(I)=ROOT;
            CALL BACKTRACK;
            END;
      END;
      END;

       /* INTERNAL SUBROUTINES */

       BACKTRACK: PROC;
          USED(I)='0'B;
          LEVEL=LEVEL-1;
          IF LEVEL¬=0 THEN I=PATH(LEVEL);
        END BACKTRACK;

        EXTEND_PATH: PROC;
          USED(J)='1'B;
          REACHJ(I)=J+1;
          LEVEL=LEVEL+1;
          PATH(LEVEL)=J;
          I=J;
        END EXTEND_PATH;
```

```
PRCYCLE:  PROC(PATH,LEVEL);
    /*** MOST OF THIS PROCEDURE HAS BEEN CHANGED FROM ADAM RHIN'S
    THESIS TO SIMPLIFY THE PRINTING OF CIRCULAR DEFINITONS
    IN THE ERROR REPORT              RB 10/76        ***/

    DCL IA FIXED;
   DCL $TST CHAR(12) EXT VARYING;
    DCL PATH(*) FIXED BIN;
    DCL LEVEL  FIXED BIN;
    DCL ERR_CNT FIXED BIN EXT;
       PRINT_LINE = 'ERROR (CIRCUULAR DEFINITION): '
       ||'THE FOLLOWING GROUP  OF ITEMS IN TEST ' || $TST
       ||' ARE CIRCULARLY DEFINED: ';
    DO NODES=1 TO LEVEL;
    PRINT-LINE=PRINT_LINE || DICT(PATH(NODES)) ||', ' ;
    END;
     IA=LENGTH(PRINT_LINE);
     PRINT_LINE=SUBSTR(PRINT_LINE,1,IA-2);
     PUT FILE (SEQERR) EDIT (PRINT_LINE) (SKIP,A);
END PRCYCLE;
END CYCLES;



PRECEED:PROC(ADJMAT,ORDER,N);
/* REARRANGE NODES OF A DIRECT GRAPH(SPECIFIED BY ADJACENCY MATRIX    */
/* "ADJMAT",N BY N) IN ASCENDING "RANK" ORDER, RESULTING IN N-ELEMENT*/
/* VECTOR "ORDER".                                                    */
  DCL (ADJMAT(*,*), UNUSE(N), T INIT('1'B), F INIT('0'B) ) BIT(1),
      (ORDER(*),NODES(2)INIT((2)0),NEW INIT(2),OLD INIT(1))FIXED BIN;
  DCL (          DEPTH(2,N), K INIT(0) ) FIXED BIN;
  DCL RANK(NR) FIXED BIN EXT CTL;
  DCL (I,J,L,II,M) FIXED BIN STATIC;

/* ADJMAT--ADJACENCY MATRIX DEFINING THE DIGRAPH.                     */
/* N-------TNE NUMBER OF NODES IN THE DIGRAPH.                        */
/* ORDER---THE VECTOR OF NODES IN RANK ORDER.                         */
/* RANK----VECTOR OF RANKS OF NODES IN DIGRAPH.   INTUITIVELY, THE    */
/*         RANK IS THE MAX. DEPTH OF A GIVEN NODE IN THE DIGRAPH      */
/*         FROM ANY ROOT. EXT BECAUSE USED IN 'GFLTRPT'               */
/* DEPTH---SET OF NODES IN A GIVEN RANK(OLD & NEW), I.E., "RANK SET",*/
/* NODES---COUNTERS FOR # OF NODES IN OLD & NEW RANK SET(DEPTH).      */
/* OLD-----POINTER TO PREVIOUS RANK SET.                              */
/* NEW-----POINTER TO CURRENT RANK SET.                               */
/* UNUSE---BIT VECTOR OF NODES NOT IN THE CURRENT RANK SET.           */

/* ALLOCATE RANK AND INITIALIZE RANK OF ALL NODES TO 0  */

    NR=N;
    ALLOCATE RANK;

    IF N=1 THEN DO; ORDER(1),RANK(1)=0; RETURN; END;

    RANK, ORDER = 0;
  /*  SET UP DEPTH 0.  */
   DO J=1 TO N;
    /* ENTER THOSE NODES WITH AN ALL-0 COLUMN IN THE FIRST RANK SET */
      IF ANY(ADJMAT(*,J)) THEN GOTO OUT;
      M, NODES(OLD)=NODES(OLD)+1;    DEPTH(OLD,M)=J;
    OUT: END;
```

```
      IF NODES(OLD)<=0 THEN GOTO ERR;       /* NO COL HAS ALL 0 */
                                            /* WHICH MEANS T8AT EVERYTHING IS
                                               DEPENDENT ON SOMETHING ELSE */

   /* OTHERWISE, PROCEED TO FIND RANK SETS OF DEPTH 1 AND ON */

     DO L=1 TO N-1;     /* FOR EACH RANK SET ("DEPTH")  */
       /* INITIALIZE NUMBER OF NODES IN NEXT RANK SET TO 0;
          FLAG ALL NODES AS NOT APPEARING IN NEXT RANK SET INITIALLY */
       NODES(NEW)=0;    UNUSE=T;

       DO I=1 TO NODES(OLD);    II=DEPTH(OLD,I);   /* FOR EACH NODE IN THE
                                                      PREVIOUS RANK SET */

         DO J=1 TO N;     /* FOR EACH COLUMN (NODE) CHECK IF IT IS A
                             DEPENDENT OF NODE IN OLD RANK SET   */

           /* IF NODE IS DEPENDENT OF CURRENT NODE IN OLD RANK SET
              (NODE II)  AND IF IT IS NOT YET IN NEXT (NEW) RANK SET
              THEN ENTER IT AS A MEMBER OF THE NEXT RANK SET */
           IF ADJMAT(II,J) THEN IF UNUSE(J) THEN
             DO; RANK(J)=L;   UNUSE(J)=F; /* SET OR UPDATE RANK OF NODE

                                  AND INDICATE THAT IT IS NEW RANK SET*/
               M, NODES(NEW)=NODES(NEW)+1;   DEPTH(NEW,M)=J;
             END;
         END;
       END;

       /* IF THERE ARE NOT ANY NODES IN NEXT RANK SET, I.E. THERE
             ARE NO NODES DEPENDENT ON ANY NODES IN PREVIOUS RANK SET,
             THEN WE ARE DONE BECAUSE EVERY NODE HAS ITS RANK */
       IF NODES(NEW)<=0 THEN GOTO REORDER;

       /* EXCHANGE OLD AND NEW RANK SETS, WHICH HAS THE EFFECT OF
          MAKING NEW RANK SET THE OLD ONE, AND A NEW "NEW" RANK SET
          WILL BE CREATED IN NEXT PASS */
       M=NEW;   NEW=OLD;   OLD=M;
     END;


   ERR:
       RETURN;  /*CYCLES EXIST.  ERROR RETURN WITH ORDER=0 */


   REORDER:  /* SORT NODES BY ASCENDING RANK ORDER.  */
     DO I=0 TO L-1;
       DO J=1 TO N;
         IF RANK(J)=I THEN DO;  K=K+1;   ORDER(K)=J;  END;
     END;
   END;
 END PRECEED;
```

APPENDIX B

```
        /*******************************************************/
        /*                                                     */
        /*  NOPAL TEST SPECIFICATION FOR RADIOSET              */
        /*                                                     */
        /*******************************************************/

    NOPAL SPECIFICATION RADIOSET;

        /*******************************************************/
        /*                                                     */
        /*  TEST MODULES:     13                               */
        /*                                                     */
        /*******************************************************/


      TEST 1;

        /* NULL STIMULI */

        /* NULL MEASUREMENT */

        LOGIC $LOGIC0010(1): *1;

          DIAGNOSIS 1:
              OPERATOR MESSAGE:
                  TYPE=#1,
              RESPONSE=?;

      TEST 2;

        /* NULL STIMULI */

        MEASUREMENT $M_2(2);

            CONJUNCTION $M_W0001($M_2):
                (<XJ24_B, XJ24_C>  = CONST_R(MRES OHM ))
                        TARGET: MRES;

            ASSERTION $M_W0002($M_2):
                MRES >  100
                        SOURCE: MRES;

        LOGIC $LOGIC0010(2): ?1, !¬2, *3;

         /*** FOLLOWING DIAGNOSIS ALREADY DEFINED BEFORE:

          DIAGNOSIS 1:
              OPERATOR MESSAGE:
                  TYPE=#1,
              RESPONSE=?;
                                                                     ***/

          DIAGNOSIS 2:
              OPERATOR MESSAGE:
                  AFFECTED COMPONENTS=INPUT_SHORT,
                  TYPE=#4;

          DIAGNOSIS 3:
              OPERATOR MESSAGE:
```

```
            OTHER PARAMETERS=(MRES, 'OHMS'),
            TYPE=D;

TEST SYTEST0003;

    STIMULI 2(SYTEST0003);

       CONJUNCTION $S_W0001(2):
          (<J24_B, GND>  = CONST_S(27.5 VOLT ));

    MEASUREMENT $M_SYTEST000(SYTEST0003);

       CONJUNCTION $M_W0001($M_SYTEST000):
           (<J22, GND>  = SINE_D(V1 VOLT ,F1 HZ ,VAR1 SEC ))
                 TARGET: F1, V1
                 SOURCE: VAR1;

       ASSERTION $M_W0002($M_SYTEST000):
           IF VAR1=60 THEN
               F1 =  5*1E+06 +- 60
           ELSE
               F1 =  5*1E+06 +- 2.5
                 SOURCE: VAR1, F1;

    LOGIC $LOGIC0010(SYTEST0003): *4, |¬5, *6;

      DIAGNOSIS 4:
          OPERATOR MESSAGE:
              TYPE=#5,
              TIME= 0.00000E+00SEC,
          RESPONSE=(VAR1);

      DIAGNOSIS 5:
          OPERATOR MESSAGE:
              AFFECTED COMPONENTS=FREQ_TOL(STD_5MHZ_FRE),
              OTHER PARAMETERS=('FREQ'),
              TYPE=#6;

      DIAGNOSIS 6:
          OPERATOR MESSAGE:
              OTHER PARAMETERS=(F1, 'HZ'),
              TYPE=D;

TEST 4;

    /* NULL STIMULI */

    MEASUREMENT $M_4(4);

       ASSERTION $M_W0001($M-4):
           V1 =  0.26 +- 0.06
                 SOURCE: V1;

    LOGIC $LOGIC0010(4): *7, |¬8;

      DIAGNOSIS 7:
          OPERATOR MESSAGE:
              OTHER PARAMETERS=(V1, ' VRMS'),
              TYPE=D;

      DIAGNOSIS 8:
          OPERATOR MESSAGE:
              AFFECTED COMPONENTS=AMPL_TOL(STD_5MHZ_FRE),
                 OTHER PARAMETERS =(' AMPL '),
TEST 5:          TYPE = #6;

   STIMULI 3(5):

      CONJUNCTION $S_W0001(3):
          (<J24_B, GND>  = CONST_S(27.5 VOLT )) &
          (<J16>  = SIGNAL_AM(2.001 MHZ ,-95 DB ,0 % ,1 KHZ ));

   MEASUREMENT $M_5(5):

      CONJUNCTION $M_W0001($M_5):
          (<XJ19_A, GND>  = SINE_D(V1 VOLT ,*,3 SEC ))
                   TARGET: V1;

      ASSERTION $M_W0002($M_5):
          V1 <= 0.7
                   SOURCE: V1;

   LOGIC $LOGIC0010(5): *9, !¬10;

     DIAGNOSIS 9:
         OPERATOR MESSAGE:
             OTHER PARAMETERS=(V1, 'VAC'),
             TYPE=D;

     DIAGNOSIS 10:
         OPERATOR MESSAGE:
             AFFECTED COMPONENTS=MIN_OUTPUT(AUDIO_10MW),
             TYPE=#8;

TEST 6:

   STIMULI $S_6(6):

      CONJUNCTION $S_W0001($S_6):
          (<J24_B, GND>  = CONST_S(27.5 VOLT )) &
          (<J16>  = SIGNAL_AM(2.001 MHZ ,-95 DB ,0 % ,1 KHZ ));

   MEASUREMENT $M_6(6):

      CONJUNCTION $M_W0001($M_6):
          (<XJ19_L, GND>  = SINE_D(V1 VOLT ,*,3 SEC ))
                   TARGET: V1;

      ASSERTION $M_W0002($M-6):
          V1 <= 0.7
                   SOURCE: V1;

   LOGIC $LOGIC0010(6): *11, !¬12;

     DIAGNOSIS 11:
         OPERATOR MESSAGE:
             OTHER PARAMETERS=(V1, ' VAC'),
             TYPE=D;

     DIAGNOSIS 12:
         OPERATOR MESSAGE:
             AFFECTED COMPONENTS=MIN_OUTPUT(AUDIO_2W),
             TYPE=#9;
```

```
TEST 7;
  STIMULI $S_7(7);

    CONJUNCTION $S_W0001($S_7):
        (<J24_3, GND>  = CONST_S(27.5 VOLT )) &
        (<J16>  = SIGNAL_AM(2.001 MHZ ,-95 DB ,0 % ,1 KHZ ));;

  MEASUREMENT $M_7(7);

    CONJUNCTION ANY_NAME($M_7):
        (<XJ19_A, GND>  = SINE_D(V1 VOLT ,*,3 SEC ))
                TARGET: V1;

    ASSERTION $M_W0002($M_7):
        V1 >= 2.33
                SOURCE: V1;

  LOGIC $LOGIC0010(7): *13, *14, !¬15;

   DIAGNOSIS 13:
       OPERATOR MESSAGE:
           TYPE=#10,
           TIME= 0.00000E+00SEC,
       RESPONSE=?;

   DIAGNOSIS 14:
       OPERATOR MESSAGE:
           OTHER PARAMETERS=(V1, ' VAC'),
           TYPE=D;

   DIAGNOSIS 15:
       OPERATOR MESSAGE:
           AFFECTED COMPONENTS=MAX_OUTPUT(AUDIO_10MW),
           OTHER PARAMETERS=(' 10 MW'),
           TYPE=#11;

TEST 8;

  STIMULI $S_8(8);

    CONJUNCTION $S_W0001($S_8):
        (<J24_B, GND>  = CONST_S(27.5 VOLT )) &
        (<J16>  = SIGNAL_AM(2.001 MHZ ,-95 DB ,0 % ,1 KHZ ));

  MEASUREMENT $M_8(8);

    CONJUNCTION $M_W0001($M_8):
        (<XJ19_L, GND>  = SINE_D(V1 VOLT ,*,3 SEC ))
                TARGET: V1;

    ASSERTION A1($M_8):
        LOG =  20*LOG10(V1/3.981E-06)
                TARGET: LOG
                SOURCE: V1;

    ASSERTION A2($M_8):
        V1 >= 35.2
                SOURCE: V1;

  LOGIC $LOGIC0010(8): *16, *17, !¬18;

   DIAGNOSIS 16:
```

```
            OPERATOR MESSAGE:
                TYPE=#12,
                TIME= 0.00000E+00SEC,
            RESPONSE=?;

        DIAGNOSIS 17:
            OPERATOR MESSAGE:
                OTHER PARAMETERS=(V1, ' VAC', LOG, 'D6'),
                TYPE=0;

        DIAGNOSIS 18:
            OPERATOR MESSAGE:
                AFFECTED COMPONENTS=MAX_OUTPUT(AUDIO_2W),
                OTHER PARAMETERS=('2W'),
                TYPE=#11;

  TEST 9:

      STIMULI $S_9(9):

          CONJUNCTION $S_W0001($S_9):
              (<J24_B, GND>  = CONST_S(27.5 VOLT )) &
              (<J16>  = SIGNAL_AM(F1 MHZ ,-95 DB ,0 % ,1 KHZ ))
                      SOURCE: F1;

          ASSERTION $S_W0002($S_9):
              F1 =  INT((4*(RANDOM+K)-2)*1000)/10000+1E-09
                      TARGET: F1
                      SOURCE: K;

          ASSERTION ASSERTION#2($S_9):
              F2 =  F1-0.001
                      TARGET: F2
                      SOURCE: F1;

          ASSERTION LOOP($S_9):
              K =  LOOP_OF(1,7,1)
                      TARGET: K;

      MEASUREMENT $M_9(9);

          CONJUNCTION $M_W0001($M_9):
              (<XJ19_L, GND>  = SINE_D(V2(K),*,0 SEC ))
                      TARGET: V2(K)
                      SOURCE: K;

          ASSERTION $M_W0002($M_9):
              LOG =  20*LOG10(V2(K)/3.981E-06)
                      TARGET: LOG
                      SOURCE: K, V2(K);

      LOGIC $LOGIC0010(9): *19, *20, *21;

        DIAGNOSIS 19:
            OPERATOR MESSAGE:
                OTHER PARAMETERS=(F2),
                TYPE=#14,
                TIME= 0.00000E+00SEC,
            RESPONSE=?;

        DIAGNOSIS 20:
            OPERATOR MESSAGE:
                OTHER PARAMETERS=(F1, ' MHZ'),
          TYPE=D;

    DIAGNOSIS 21:
        OPERATOR MESSAGE:
            OTHER PARAMETERS=(V2(K), ' VAC', LOG, ' DB'),
            TYPE=D;

TEST 10;

   /* NULL STIMULI */

   MEASUREMENT $M_10(10);

      ASSERTION $M_W0001($M_10):
          RATIO =  MAX(V2)/MIN(V2)
                  TARGET: RATIO
                  SOURCE: V2;

      ASSERTION $M_W0002($M_10):
          RATIO <= 4
                  SOURCE: RATIO;

   LOGIC $LOGIC0010(10): *22, !¬23;

     DIAGNOSIS 22:
        OPERATOR MESSAGE:
            OTHER PARAMETERS=(RATIO),
            TYPE=D;

     DIAGNOSIS 23:
        OPERATOR MESSAGE:
            AFFECTED COMPONENTS=GAIN_RATIO,
            TYPE=#13;

TEST 11;

   STIMULI 6(11);

      CONJUNCTION $S_W0001(6):
          (<J24_B, GND>  = CONST_S(27.5 VOLT )) &
          (<J16>  = SIGNAL_AM(25.002 MHZ ,+13 DB ,0 % ,1 KHZ ));

   MEASUREMENT $M_11(11);

      CONJUNCTION $M_W0001($M_11):
          (<XJ19_A, GND>  = SINE_O(V1 VOLT ,*,0 SEC ))
                  TARGET: V1;

      ASSERTION #1($M_11):
          V1 >= 2.2
                  SOURCE: V1;

      ASSERTION #2($M_11):
          V1 <= 2.8
                  SOURCE: V1;

   LOGIC $LOGIC0010(11): *24, *25, !¬26;

     DIAGNOSIS 24:
        OPERATOR MESSAGE:
            TYPE=#15,
            TIME= 0.00000E+00SEC,
        RESPONSE=?;
      DIAGNOSIS 25:
          OPERATOR MESSAGE:
              OTHER PARAMETERS=(V1, ' VAC'),
              TYPE=D;

      DIAGNOSIS 26:
          OPERATOR MESSAGE:
              TYPE=#17;

TEST 12;

   STIMULI DUMMY_NAME(12);

      CONJUNCTION $S_W0001(DUMMY_NAME):
          (<J24_B, GND>  = CONST_S(27.5 VOLT )) &
          (<J16>  = SIGNAL_AM(25.002 MHZ ,+13 DB ,0 % ,1 KHZ ));

   MEASUREMENT $M_12(12);

      CONJUNCTION $M_W0001($M_12):
          (<XJ19_A, GND>  = DISTORTION(P1 % ,2 KHZ ))
                    TARGET: P1;

      ASSERTION $M_W0002($M-12):
          P1 <= 3
                    SOURCE: P1;

   LOGIC $LOGIC0010(12): *27, |¬28;

      DIAGNOSIS 27:
          OPERATOR MESSAGE:
              OTHER PARAMETERS=(P1, '%'),
              TYPE=D;

      DIAGNOSIS 28:
          OPERATOR MESSAGE:
              AFFECTED COMPONENTS=DISTORT(AUDIO_10MW),
              OTHER PARAMETERS=('10 MW', 1.0),
              TYPE=#18;
TEST 13;

   STIMULI $S_13(13);

      CONJUNCTION $S_W0001($S_13):
          (<J24_B, GND>  = CONST_S(27.5 VOLT )) &
          (<J16>  = SIGNAL_AM(25.002 MHZ ,+13 DB ,0 % ,1 KHZ ));

   MEASUREMENT $M_13(13);

      CONJUNCTION $M_W0001($M_13):
          (<XJ19_L, GND>  = DISTORTION(P1 % ,1 KHZ ))
                    TARGET: P1;

      ASSERTION $M_W0002($M-13):
          P1 <= 5
                    SOURCE: P1;

   LOGIC $LOGIC0010(13): *29, |¬30;

      DIAGNOSIS 29:
          OPERATOR MESSAGE:
                OTHER PARAMETERS = (P1, '%'),
                TYPE=D;

        DIAGNOSIS 30:
            OPERATOR MESSAGE:
                AFFECTED COMPONENTS=DISTORT(AUDIO_2W),
                OTHER PARAMETERS=(' 2W', 5.0),
                TYPE=#18;

        /***********************************************/
        /*                                             */
        /*  MESSAGES                                   */
        /*                                             */
        /***********************************************/

    MESSAGE #4:
        TEXT=' R/T DC INPUT SHORTED J24-B/J24-C
                  AN/GRC-106 DEFECTIVE
                  CHECK PRINTOUTS FOR DEFECTS.
                       PRESS STOP. ';

    MESSAGE D:  ALIAS=DISPLAY,
        TEXT=' ($TETS):  (P) ';

    MESSAGE #5:
        TEXT='IF A 12 MINUTE UUT WARMUP IS DESIRED, KEY IN 720;
               OTHERWISE KEY IN 60.
                 PRESS YES. ';

    MESSAGE #6:
        TEXT='TEXT OMITTED.';

    MESSAGE #8:
        TEXT='TEXT OMITTED.';

    MESSAGE #9:
        TEXT='TEXT OMITTED.';

    MESSAGE #10:
        TEXT='TEXT OMITTED.';

    MESSAGE #11:
        TEXT='TEXT OMITTED.';

    MESSAGE #12:
        TEXT='TEXT OMITTED.';

    MESSAGE #14:
        TEXT='TEXT OMITTED.';

    MESSAGE #13:
        TEXT='TEXT OMITTED.';

    MESSAGE #15:
        TEXT='TEXT OMITTED.';

    MESSAGE #17:
        TEXT='TEXT OMITTED.';

    MESSAGE #18:
        TEXT=' (P1) AUDIO DISTORTION GREATER THAN (P2) PERCENT. ';
```

```
MESSAGE 1:  ALIAS=#1,
    TEXT=' INITIAL UUT & ATE  HOOKUP MESSAGE.   FULL TEXT OMITTED. ';

  /**********************************************/
  /*                                            */
  /*  UUT COMPONENTS/FAILURES                   */
  /*                                            */
  /**********************************************/


COMP_FAIL 1: INPUT_SHORT;

COMP_FAIL 2: STD_5MHZ_FRE, FAILURE FUNCTION=FREQ_TOL, INDEX=1,PROTECT=(1);

COMP_FAIL 3: STD_5MHZ_FRE, FAILURE FUNCTION=AMPL_TOL, INDEX=2,PROTECT=(1);

COMP_FAIL 4: AUDIO_10MW, FAILURE FUNCTION=MIN_OUTPUT, INDEX=3,PROTECT=(1)

COMP_FAIL 5: AUDIO_10MW, FAILURE FUNCTION=MAX_OUTPUT, INDEX=5,PROTECT=(1)'

COMP_FAIL 6: AUDIO_10MW, FAILURE FUNCTION=REF_VOLT, PROTECT=(1, 11),
        COMMENTS=' DISTORTION REF. VOLT. TEST.';

COMP_FAIL 11: GAIN_RATIO, PROTECT=(1, 2, 3, 4, 5, 8, 9);

COMP_FAIL 7: AUDIO_10MW, FAILURE FUNCTION=DISTORT, PROTECT=(1, 6);

COMP_FAIL 8: AUDIO_2W, FAILURE FUNCTION=MIN_OUTPUT, INDEX=4, PROTECT=(

COMP_FAIL 9: AUDIO_2W, FAILURE FUNCTION=MAX_OUTPUT, INDEX=6, PROTECT=(

COMP_FAIL 10: AUDIO_2W, FAILURE FUNCTION=DISTORT, PROTECT=(1, 11);

  /**********************************************/
  /*                                            */
  /*  UUT CONNECTION POINTS                     */
  /*                                            */
  /**********************************************/


UUT_POINT       : XJ24_C, ALIAS=GND, CONNECTOR=(MULTIPLE, C);

UUT_POINT       : J24_B, ALIAS=XJ24_B, CONNECTOR=(MULTIPLE, B),
        LIMIT=(VOLT,  3.50000E+01,  2.00000E+01, GND),
        COMMENTS=' MULTIPLE CONNECTOR';

UUT_POINT       : J22;

UUT_POINT      3: J16, ALIAS=XJ16, CONNECTOR=(COAX, ),
        LIMIT=(UVOLT,  1.00000E+02,  0.00000E+00, GND),
        COMMENTS=' COAXIAL CABLE';

UUT_POINT      2: J24_C, ALIAS=GND, CONNECTOR=(MULTIPLE, C);

UUT_POINT     40: J19_A, ALIAS=XJ19_A, CONNECTOR=(MULTIPLE, A),
        LIMIT=(VOLT,  5.00000E+00,  0.00000E+00, GND);

UUT_POINT     50: J19_B, ALIAS=GND, CONNECTOR=(MULTIPLE, B);

UUT_POINT       : J19_L, ALIAS=XJ19_L, CONNECTOR=(MULTIPLE, L),
        LIMIT=(,  7.00000E+01,  0.00000E+00, GND);

  /**********************************************/
```

```
       /*  ATE FUNCTIONS                                   */
       /*                                                  */
       /***************************************************/

FUNCTION    20: CONST_R, FUNCTION TYPE=M, #PINS= 2,
        PARAM_01=(X, T, LIMIT=(OHM,  1.00000E+03,  1.00000E+00)),
        VALUE RETURNED='TRUE/FALSE';

FUNCTION    10: CONST_S, FUNCTION TYPE=S, #PINS= 2,
        PARAM_01=(X, S, LIMIT=(VOLT,  6.00000E+01,  0.00000E+00)),
        VALUE RETURNED=' CONSTANT VOLT.';

FUNCTION    30: SINE_D, ALIAS=SINE_DELAY, FUNCTION TYPE=M, #PINS= 2,
        PARAM_01=(X, T, LIMIT=(VOLT,  1.00000E+01, -1.00000E+01)),
        PARAM_02=(Y, T, LIMIT=(MHZ,  1.00000E+01,  0.00000E+00)),
        PARAM_03=(Z, S, LIMIT=(SEC,  1.00000E+75, -1.00000E+75)),
        COMMENTS='APMPL., FREQ., TIME DELYD';

FUNCTION   110: FREQ_TOL, FUNCTION TYPE=F,
        PARAM_01=(COMPONENT, S);

FUNCTION   120: AMPL_TOL, FUNCTION TYPE=F,
        PARAM_01=(COMPONENT, S);

FUNCTION   130: MIN_OUTPUT, FUNCTION TYPE=F,
        PARAM_01=(COMPONENT, S);

FUNCTION   140: MAX_OUTPUT, FUNCTION TYPE=F,
        PARAM_01=(COMPONENT, S);

FUNCTION      : DISTORT, FUNCTION TYPE=F,
        PARAM_01=(COMPONENT, S);

FUNCTION   150: REF_VOLT, FUNCTION TYPE=F,
        PARAM_01=(COMPONENT, S);

FUNCTION    40: SAM, ALIAS=SIGNAL_AM, FUNCTION TYPE=S, #PINS= 1,
        PARAM_01=(X, S, LIMIT=(MHZ,  1.00000E+02,  1.00000E-01)),
        PARAM_02=(Y, S, LIMIT=(DB, -1.00000E+01, -1.50000E+02)),
        PARAM_03=(Z, S, LIMIT=(%,  1.00000E+75, -1.00000E+75)),
        PARAM_04=(W, S, LIMIT=(KHZ,  1.50000E+01,  1.00000E-01));

FUNCTION    50: LOG10, FUNCTION TYPE=E,
        PARAM_01=(X, S),
        VALUE RETURNED='LOG10(X)';

FUNCTION    60: INT, ALIAS=INTEGER, FUNCTION TYPE=E,
        PARAM_01=(X, S),
        VALUE RETURNED='FLOOR OF X', COMMENTS=' LARGEST INTEGER <= X';

FUNCTION    70: RAND, ALIAS=RANDOM, FUNCTION TYPE=E,
        VALUE RETURNED='RANDOM NUMBER';

FUNCTION    80: MAX, ALIAS=MAXIMUM, FUNCTION TYPE=E,
        VALUE RETURNED='MAX(X1,X2, XN)', COMMENTS=' N>=1: X A VECTOR.'

FUNCTION    90: MIN, ALIAS=MINIMUM, FUNCTION TYPE=E,
        VALUE RETURNED=' MIN(X1,X2,XN)';

FUNCTION      : DIST, ALIAS=DISTORTION, FUNCTION TYPE=M, #PINS= 2,
        PARAM_01=(X, T, LIMIT=(%,  1.00000E+75, -1.00000E+75)),
            PARAM_02=(Y, S, LIMIT=(KHZ,  1.00000E+02,  0.00000E+00)),
            VALUE RETURNED='TRUE/FALSE';

     FUNCTION     : LOOP_OF, FUNCTION TYPE=C,
            PARAM_01=(INIT, S),
            PARAM_02=(BOUND, S),
            PARAM_03=(INCR, S), COMMENTS='REPEAT FOR K=INIT TO BOUND BY INCR'

        /*************************************************/
        /*                                               */
        /*  ATE CONNECTION POINTS                        */
        /*                                               */
        /*************************************************/

     ATE_POINT      1: ATE_XJ24B, UUT_POINTS=(J24_B);

     ATE_POINT       : ATE_XJ24_C, UUT_POINTS=(J24_C);
     .
     ATE_POINT '    30: ATEPT#30, UUT_POINTS=(J16, J22);

   END RADIOSET;
```

APPENDIX C

```
                        INTRA MODULE SEQUENCING 1
                        ANALYSIS OF THE ADJACENCY MATRIX
                                      1
                                      -
   1  1               DIAGNOSES       0

                        SEQUENCE OF PROCESSING FOR TEST 1
ORDER
VECT  ORDER   RANK        NAME           TYPE                    TEXT
INDEX VECTOR
  1     0      1          1            DIAGNOSES     TYPE = #1,  RESPONSE = ?!
```

```
                          INTRA MODULE SEQUENCING 2
                          ANALYSIS OF THE ADJACENCY MATRIX
                                           1 2 3 4 5 6
                                           - - - - - -
     1  $M_W0001       CONJUNCTION         0 0 0 0 0 1
     2  $M_W0002       ASSERTION           0 0 0 0 0 0
     3  1              DIAGNOSES           0 0 0 0 0 0
     4  2              DIAGNOSES           0 0 0 0 0 0
     5  3              DIAGNOSES           0 0 0 0 0 0
     6  MRES           VARIABLE            0 1 0 0 1 0

                          SEQUENCE OF PROCESSING FOR TEST 2
ORDER
VECT  ORDER   RANK       NAME            TYPE                     TEXT
INDEX VECTOR
  1     1      0        $M_W0001        CONJUNCTION       (<XJ24_B, XJ24_C>  = CONST_R(MRES OHM ))
                                                          TARGET:  MRES;

  2     3      0        1               DIAGNOSES         TYPE = #1,  RESPONSE = ?;

  3     4      0        2               DIAGNOSES         AFFECTED COMPONENTS = INPUT_SHORT,  TYPE = #4;

  4     6      1        MRES            VARIABLE          LOCAL

  5     2      2        $M_W0002        ASSERTION         MRES >  100
                                                          SOURCE:  MRES;

  6     5      2        3               DIAGNOSES         OTHER PARAMETERS=('OHMS', MRES),  TYPE = D;
```

```
                         INTRA MODULE SEQUENCING SYTEST0003
                         ANALYSIS OF THE ADJACENCY MATRIX
                                       1 2 3 4 5 6 7 8 9
                                       - - - - - - - - -
  1  $S_W0001          CONJUNCTION     0 1 0 0 0 0 0 0 0
  2  $M_W0001          CONJUNCTION     0 0 0 0 0 0 0 1 1
  3  $M_W0002          ASSERTION       0 0 0 0 0 0 0 0 0
  4  4                 DIAGNOSES       0 0 0 0 0 0 1 0 0
  5  5                 DIAGNOSES       0 0 0 0 0 0 0 0 0
  6  6                 DIAGNOSES       0 0 0 0 0 0 0 0 0
  7  VAR1              VARIABLE        0 1 1 0 0 0 0 0 0
  8  F1                VARIABLE        0 0 1 0 0 1 0 0 0
  9  V1                VARIABLE        0 0 0 0 0 0 0 0 0


                         SEQUENCE OF PROCESSING FOR TEST SYTEST0003
ORDER
VECT  ORDER   RANK       NAME             TYPE                     TEXT
INDEX VECTOR
  1     1      0         $S_W0001         CONJUNCTION        (<J24_B, GND>  = CONST_S(27.5 VOLT ));

  2     4      0         4                DIAGNOSES          TYPE = #5, TIME= 0.00000E+00,  RESPONSE=( VAR1);

  3     5      0         5                DIAGNOSES          AFFECTED COMPONENTS = FREQ_TOL(STD_5MHZ_FRE), OTHER PARAMETERS=(
                                                             'FREQ'),  TYPE = #6;

  4     7      1         VAR1             VARIABLE           LOCAL

  5     2      2         $M_W0001         CONJUNCTION        (<J22, GND>  = SINE_D(V1 VOLT ,F1 HZ ,VAR1 SEC ))
                                                             TARGET:  F1,  V1
                                                             SOURCE:  VAR1;

  6     8      3         F1               VARIABLE           LOCAL

  7     9      3         V1               VARIABLE           GLOBAL / TARGET /

  8     3      4         $M_W0002         ASSERTION          IF VAR1=60 THEN F1 =  5*1E+06 +- 60
                                                             ELSE F1 =  5*1E+06 +- 2.5
                                                             SOURCE:  VAR1,  F1;

  9     6      4         6                DIAGNOSES          OTHER PARAMETERS=('HZ', F1),  TYPE = D;
```

```
                         INTRA MODULE SEQUENCING 4
                         ANALYSIS OF THE ADJACENCY MATRIX
                                      1 2 3 4
                                      - - - -
   1  $M_W0001          ASSERTION     0 0 0 0
   2  7                 DIAGNOSES     0 0 0 0
   3  8                 DIAGNOSES     0 0 0 0
   4  V1                VARIABLE      1 1 0 0

                         SEQUENCE OF PROCESSING FOR TEST 4
 ORDER
 VECT  ORDER   RANK        NAME  .        TYPE                     TEXT
 INDEX VECTOR
   1     3      0          8             DIAGNOSES      AFFECTED COMPONENTS = AMPL_TOL(STD_5MHZ_FRE), OTHER PARAMETERS=
                                                        ' AMPL'),  TYPE = #6;

   2     4      0          V1            VARIABLE       GLOBAL / SOURCE /

   3     1      1          $M_W0001      ASSERTION      V1 =  0.26 +- 0.06
                                                        SOURCE:  V1;

   4     2      1          7             DIAGNOSES      OTHER PARAMETERS=(' VRMS', V1),  TYPE = 0;
```

AD-A053 315 MOORE SCHOOL OF ELECTRICAL ENGINEERING PHILADELPHIA P--ETC F/G 14/2
NOPAL PROCESSOR: INTRA-TEST SEQUENCING.(U)
JAN 78 R W BERMAN DAAA25-75-C-0650
UNCLASSIFIED ECOM-75-0650-F NL

2 OF 2
AD
A053315

END
DATE
FILMED
6-78
DDC

```
                    INTRA MODULE SEQUENCING 5
                    ANALYSIS OF THE ADJACENCY MATRIX
                                     1 2 3 4 5 6
                                     - - - - - -
    1  $S_W0001       CONJUNCTION    0 1 0 0 0 0
    2  $M_W0001       CONJUNCTION    0 0 0 0 0 1
    3  $M_W0002       ASSERTION      0 0 0 0 0 0
    4  9              DIAGNOSES      0 0 0 0 0 0
    5  10             DIAGNOSES      0 0 0 0 0 0
    6  V1             VARIABLE       0 0 1 1 0 0


                    SEQUENCE OF PROCESSING FOR TEST 5
ORDER
VECT  ORDER   RANK      NAME          TYPE                          TEXT
INDEX VECTOR
  1     1      0      $S_W0001     CONJUNCTION      (<J24_B, GND>  = CONST_S(27.5 VOLT )) &(<J16>  =
                                                    SIGNAL_AM(2.001 MHZ ,-95 DB ,0 % ,1 KHZ ));

  2     5      0      10           DIAGNOSES        AFFECTED COMPONENTS = MIN_OUTPUT(AUDIO_10MW),  TYPE = #B;

  3     2      1      $M_W0001     CONJUNCTION      (<XJ19_A, GND>  = SINE_D(V1 VOLT ,*,3 SEC ))
                                                    TARGET:  V1;

  4     6      2      V1           VARIABLE         GLOBAL / TARGET /

  5     3      3      $M_W0002     ASSERTION        V1 <= 0.7
                                                    SOURCE:  V1;

  6     4      3      9            DIAGNOSES        OTHER PARAMETERS=('VAC', V1),  TYPE = D;
```

```
                            INTRA MODULE SEQUENCING 6
                            ANALYSIS OF THE AUJACENCY MATRIX
                                       1 2 3 4 5 6
                                       - - - - - -
     1  $S_W0001        CONJUNCTION    0 1 0 0 0 0
     2  $M_W0001        CONJUNCTION    0 0 0 0 0 1
     3  $M_W0002        ASSERTION      0 0 0 0 0 0
     4  11              DIAGNOSES      0 0 0 0 0 0
     5  12              DIAGNOSES      0 0 0 0 0 0
     6  V1              VARIABLE       0 0 1 1 0 0


                            SEQUENCE OF PROCESSING FOR TEST 6
ORDER
VECT  ORDER    RANK       NAME          TYPE                      TEXT
INDEX VECTOR
   1     1      0        $S_W0001      CONJUNCTION       (<J24_B, GND>  = CONST_S(27.5 VOLT )) &(<J16>  =
                                                         SIGNAL_AM(2.001 MHZ ,-95 DB ,0 % ,1 KHZ ));

   2     5      0        12            DIAGNOSES         AFFECTED COMPONENTS = MIN_OUTPUT(AUDIO_2W),  TYPE = #9;

   3     2      1        $M_W0001      CONJUNCTION       (<XJ19_L, GND>  = SINE_D(V1 VOLT ,*,3 SEC ))
                                                         TARGET:  V1;

   4     6      2        V1            VARIABLE          GLOBAL / TARGET /

   5     3      3        $M_W0002      ASSERTION         V1 <= 0.7
                                                         SOURCE:  V1;

   6     4      3        11            DIAGNOSES         OTHER PARAMETERS=(' VAC', V1),  TYPE = D;
```

```
                              INTRA MODULE SEQUENCING 7
                              ANALYSIS OF THE ADJACENCY MATRIX
                                     1 2 3 4 5 6 7
                                     - - - - - - -
      1  $S_W0001         CONJUNCTION  0 1 0 0 0 0 0
      2  ANY_NAME         CONJUNCTION  0 0 0 0 0 0 1
      3  $M_W0002         ASSERTION    0 0 0 0 0 0 0
      4  13               DIAGNOSES    0 0 0 0 0 0 0
      5  14               DIAGNOSES    0 0 0 0 0 0 0
      6  15               DIAGNOSES    0 0 0 0 0 0 0
      7  V1               VARIABLE     0 0 1 0 1 0 0

                              SEQUENCE OF PROCESSING FOR TEST 7
ORDER
VECT  ORDER   RANK      NAME          TYPE                 TEXT
INDEX VECTOR
  1     1       0       $S_W0001      CONJUNCTION    (<J24_B, GND>  = CONST_S(27.5 VOLT )) &(<J16>  =
                                                     SIGNAL_AM(2.001 MHZ ,-95 DB ,0 % ,1 KHZ ));

  2     4       0       13            DIAGNOSES      TYPE = #10, TIME= 0.00000E+00,  RESPONSE = 7;

  3     6       0       15            DIAGNOSES      AFFECTED COMPONENTS = MAX_OUTPUT(AUDIO_10MW), OTHER PARAMETERS=(
                                                     ' 10 MW'),  TYPE = #11;

  4     2       1       ANY_NAME      CONJUNCTION    (<XJ19_A, GND>  = SINE_D(V1 VOLT ,*,3 SEC ))
                                                     TARGET:  V1;

  5     7       2       V1            VARIABLE       GLOBAL / TARGET /

  6     3       3       $M_W0002      ASSERTION      V1 >= 2.33
                                                     SOURCE:  V1;

  7     5       3       14            DIAGNOSES      OTHER PARAMETERS=(' VAC', V1),  TYPE = D;
```

```
                        INTRA MODULE SEQUENCING 8
                        ANALYSIS OF THE AUJACENCY MATRIX
                                   1 2 3 4 5 6 7 8 9
                                   - - - - - - - - -
       1  $S_W0001      CONJUNCTION   0 1 0 0 0 0 0 0 0
       2  $M_W0001      CONJUNCTION   0 0 0 0 0 0 0 1 0
       3  A1            ASSERTION     0 0 0 0 0 0 0 0 1
       4  A2            ASSERTION     0 0 0 0 0 0 0 0 0
       5  16            DIAGNOSES     0 0 0 0 0 0 0 0 0
       6  17            DIAGNOSES     0 0 0 0 0 0 0 0 0
       7  18            DIAGNOSES     0 0 0 0 0 0 0 0 0
       8  V1            VARIABLE      0 0 1 1 0 1 0 0 0
       9  LOG           VARIABLE    . 0 0 0 0 0 1 0 0 0

                        SEQUENCE OF PROCESSING FOR TEST 8
ORDER
VECT  ORDER   RANK     NAME          TYPE                TEXT
INDEX VECTOR
   1     1      0     $S_W0001      CONJUNCTION     (<J24_B, GND>  = CONST_S(27.5 VOLT )) &(<J16>  =
                                                    SIGNAL_AM(2.001 MHZ ,-95 DB ,0 % ,1 KHZ ));

   2     5      0     16            DIAGNOSES       TYPE = #12, TIME= 0.00000E+00,  RESPONSE = ?;

   3     7      0     18            DIAGNOSES       AFFECTED COMPONENTS = MAX_OUTPUT(AUDIO_2W), OTHER
                                                    PARAMETERS= ('2W'), TYPE = #11;

   4     2      1     $M_W0001      CONJUNCTION     (<XJ19_L, GND>  = SINE_D(V1 VOLT ,*,3 SEC ))
                                                    TARGET:  V1;

   5     8      2     V1            VARIABLE        GLOBAL / TARGET /

   6     3      3     A1            ASSERTION       LOG =  20*LOG10(V1/3.981E-06)
                                                    TARGET:  LOG
                                                    SOURCE:  V1;

   7     4      3     A2            ASSERTION       V1 >= 35.2
                                                    SOURCE:  V1;

   8     9      4     LOG           VARIABLE        LOCAL

   9     6      5     17            DIAGNOSES       OTHER PARAMETERS=(' VAC', 'DB', V1, LOG),  TYPE = D;
```

```
                               INTRA MODULE SEQUENCING 9
                               ANALYSIS OF THE ADJACENCY MATRIX
                                                         1 1 1 1 1
                                     1 2 3 4 5 6 7 8 9 0 1 2 3 4
                                     - - - - - - - - - - - - - -
  1  $S_W0001       CONJUNCTION      0 0 0 0 1 0 0 0 0 0 0 0 0 0
  2  $S_W0002       ASSERTION        0 0 0 0 0 0 0 0 0 1 0 0 0 0
  3  ASSERTION#2    ASSERTION        0 0 0 0 0 0 0 0 0 0 0 1 0 0
  4  LOOP           ASSERTION        0 0 0 0 0 0 0 0 0 0 1 0 0 0
  5  $M_W0001       CONJUNCTION      0 0 0 0 0 0 0 0 0 0 0 0 1 0
  6  $M_W0002       ASSERTION        0 0 0 0 0 0 0 0 0 0 0 0 0 1
  7  19             DIAGNOSES        0 0 0 0 0 0 0 0 0 0 0 0 0 0
  8  20             DIAGNOSES      . 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  9  21             DIAGNOSES        0 0 0 0 0 0 0 0 0 0 0 0 0 0
 10  F1             VARIABLE         1 0 1 0 0 0 0 1 0 0 0 0 0 0
 11  K              VARIABLE         0 1 0 0 1 1 0 0 0 0 0 0 0 0
 12  F2             VARIABLE         0 0 0 0 0 0 1 0 0 0 0 0 0 0
 13  V2(K)          VARIABLE         0 0 0 0 0 1 0 0 1 0 0 0 0 0
 14  LOG            VARIABLE         0 0 0 0 0 0 0 0 1 0 0 0 0 0


                               SEQUENCE OF PROCESSING FOR TEST 9
ORDER
VECT   ORDER    RANK       NAME            TYPE                    TEXT
INDEX  VECTOR
  1      4        0        LOOP            ASSERTION       K =  LOOP_OF(1,7,1)
                                                           TARGET:  K;

  2     11        1        K               VARIABLE        LOCAL

  3      2        2        $S_W0002        ASSERTION       P1 =  INT((4*(RANDOM+K)-2)*1000)/10000+1E-09
                                                           TARGET:  F1
                                                           SOURCE:  K;

  4     10        3        F1              VARIABLE        LOCAL

  5      1        4        $S_W0001        CONJUNCTION     (<J24_B, GND>  = CONST_S(27.5 VOLT )) &(<J16>  =
                                                           SIGNAL_AM(F1 MHZ ,-95 DB ,0 % ,1 KHZ ))
                                                           SOURCE:  F1;

  6      3        4        ASSERTION#2     ASSERTION       P2 =  F1-0.001
                                                           TARGET:  F2
                                                           SOURCE:  P1;
```

```
   7     8     4      20           DIAGNOSES       OTHER PARAMETERS=(' MHZ', F1),  TYPE = 0;

   8     5     5    SM_W0001       CONJUNCTION     (<XJ19_L, GND>  = SINE_D(V2(K),*,0 SEC ))
                                                   TARGET:  V2(K)
                                                   SOURCE:  K;

   9    12     5      F2           VARIABLE        LOCAL

  10     7     6      19           DIAGNOSES       OTHER PARAMETERS=(F2),  TYPE = #14,
                                                   TIME= 0.00000E+00,  RESPONSE = ?;

  11    13     6      V2(K)        SUBSCRPT_VAR    GLOBAL / TARGET /
```

```
12     6      7        $M_W0002      ASSERTION      LOG =  20*LOG10(V2(K)/3.981E-06)
                                                    TARGET:  LOG
                                                    SOURCE:  K,  V2(K);

13    14      A        LOG           VARIABLE       LOCAL

14     9      9        21            DIAGNOSES      OTHER PARAMETERS=(' VAC', ' DB', V2(K), LOG),  TYPE = D;
```

```
                         INTRA MODULE SEQUENCING 10
                         ANALYSIS OF THE AUJACENCY MATRIX
                                             1 2 3 4 5 6
                                             - - - - - -
      1  $M_W0001         ASSERTION          0 0 0 0 0 1
      2  $M_W0002         ASSERTION          0 0 0 0 0 0
      3  22               DIAGNOSES          0 0 0 0 0 0
      4  23               DIAGNOSES          0 0 0 0 0 0
      5  V2               VARIABLE           1 0 0 0 0 0
      6  RATIO            VARIABLE           0 1 1 0 0 0

                         SEQUENCE OF PROCESSING FOR TEST 10
ORDER
VECT  ORDER   RANK        NAME           TYPE                        TEXT
INDEX VECTOR
  1     4      0         23            DIAGNOSES         AFFECTED COMPONENTS = GAIN_RATIO,  TYPE = #13;

  2     5      0         V2            VARIABLE          GLOBAL / SOURCE /

  3     1      1         $M_W0001      ASSERTION         RATIO =  MAX(V2)/MIN(V2)
                                                         TARGET:  RATIO
                                                         SOURCE:  V2;

  4     6      2         RATIO         VARIABLE          LOCAL

  5     2      3         $M_W0002      ASSERTION         RATIO <= 4
                                                         SOURCE:  RATIO;

  6     3      3         22            DIAGNOSES         OTHER PARAMETERS=(RATIO),  TYPE = D;
```

```
                    INTRA MODULE SEQUENCING 11
                    ANALYSIS OF THE ADJACENCY MATRIX
                                     1 2 3 4 5 6 7 8
                                     - - - - - - - -
      1  $S_W0001       CONJUNCTION  0 1 0 0 0 0 0 0
      2  $M_W0001       CONJUNCTION  0 0 0 0 0 0 0 1
      3  #1             ASSERTION    0 0 0 0 0 0 0 0
      4  #2             ASSERTION    0 0 0 0 0 0 0 0
      5  24             DIAGNOSES    0 0 0 0 0 0 0 0
      6  25             DIAGNOSES    0 0 0 0 0 0 0 0
      7  26             DIAGNOSES    0 0 0 0 0 0 0 0
      8  V1             VARIABLE   . 0 0 1 1 0 1 0 0

                    SEQUENCE OF PROCESSING FOR TEST 11
ORDER
VECT  ORDER   RANK      NAME        TYPE                    TEXT
INDEX VECTOR
  1     1      0      $S_W0001     CONJUNCTION     (<J24_B, GND>  = CONST_S(27.5 VOLT )) &(<J16>  =
                                                   SIGNAL_AM(25.002 MHZ ,+13 DB ,0 % ,1 KHZ ));

  2     5      0      24           DIAGNOSES       TYPE = #15, TIME= 0.00000E+00, RESPONSE = ?;

  3     7      0      26           DIAGNOSES       TYPE = #17;

  4     2      1      $M_W0001     CONJUNCTION     (<XJ19_A, GND>  = SINE_D(V1 VOLT ,*,0 SEC ))
                                                   TARGET:  V1;

  5     8      2      V1           VARIABLE        GLOBAL / TARGET /

  6     3      3      #1           ASSERTION       V1 >= 2.2
                                                   SOURCE:  V1;

  7     4      3      #2           ASSERTION       V1 <= 2.8
                                                   SOURCE:  V1;

  8     6      3      25           DIAGNOSES       OTHER PARAMETERS=(' VAC', V1),  TYPE = D;
```

```
                              INTRA MODULE SEQUENCING 12
                              ANALYSIS OF THE ADJACENCY MATRIX
                                          1 2 3 4 5 6
                                          - - - - - -
      1  $S_W0001          CONJUNCTION    0 1 0 0 0 0
      2  $M_W0001          CONJUNCTION    0 0 0 0 0 1
      3  $M_W0002          ASSERTION      0 0 0 0 0 0
      4  27                DIAGNOSES      0 0 0 0 0 0
      5  28                DIAGNOSES      0 0 0 0 0 0
      6  P1                VARIABLE       0 0 1 1 0 0

                              SEQUENCE OF PROCESSING FOR TEST 12
ORDER
VECT  ORDER   RANK     NAME          TYPE                      TEXT
INDEX VECTOR
   1     1      0      $S_W0001      CONJUNCTION      (<J24_B, GND>  = CONST_S(27.5 VOLT )) &(<J16>  =
                                                      SIGNAL_AM(25.002 MHZ ,+13 DB ,0 % ,1 KHZ ));

   2     5      0      28            DIAGNOSES        AFFECTED COMPONENTS = DISTORT(AUDIO_10MW), OTHER PARAMETERS=(
                                                      '10 MW', 1.0),  TYPE = #18;

   3     2      1      $M_W0001      CONJUNCTION      (<XJ19_A, GND>  = DISTORTION(P1 % ,2 KHZ ))
                                                      TARGET:  P1;

   4     6      2      P1            VARIABLE         LOCAL

   5     3      3      $M_W0002      ASSERTION        P1 <= 3
                                                      SOURCE:  P1;

   6     4      3      27            DIAGNOSES        OTHER PARAMETERS=('%', P1),  TYPE = D;
```

DISTRIBUTION LIST

For Technical Report No. ECOM-75-0650-F

Defense Documentation Center
ATTN: DDC-TCA
Cameron Station (Bldg. 5)
Alexandria, VA 22314 (12 cys)

Code R123, Tech Library
DCA Defense Comm Engrg Ctr
1870 Wiehle Ave.
Reston, VA 22090

Defense Communications Agency
Technical Library Center
Code 205 (P.A. Tolovi)
Washington, DC 20305

GIDEP Engineering & Support Dept.
TE Section
PO Box 398
Norco, CA 91760

Commander
Naval Electronics Laboratory Center
ATTN: Library
San Diego, CA 92152

Rome Air Development Center
ATTN: Documents Library (TILD)
Griffiss AFB, NY 13441

CDR, US Army Missile Command
Redstone Scientific Info Center
ATTN: Chief, Document Section
Redstone Arsenal, AL 35809 (2 cys)

Commander
HQ Fort Huachuca
ATTN: Technical Reference Div.
Fort Huachuca, AZ 85613

Commandant
US Army Signal School
ATTN: ATSN-CTD-MS
Fort Gordon, GA 30905

CDR, Harry Diamond Laboratories
ATTN: Library
2800 Powder Mill Road
Adelphi, MD 20783

HQ Marine Corps
CODE LMC-4
Washington, DC 20380

Director, National Security Agency
NSA/S274
ATTN: Mr. Ron Seaman
Fort Meade, MD 20755

Commander
TRASANA
ATTN: ATAA-TDL
White Sands Missile Range, NM 88002

DA-ODCSLOG
ATTN: DALO-SMM-E (Mr. Nichols)
Washington, DC 20310

ASA L&L
ATTN: COL E. A. Viereck, Jr.
Pentagon Building
Room 3E619
Washington, DC 20310

PM, ATSS
ATTN: DRCPM-ATSS
Fort Monmouth, NJ 07703

Commander
US Army Troop Support Command
4300 Goodfellow Boulevard
St. Louis, MO 63120

Commander
ARRCOM
ATTN: DRSAR-RDG
Rock Island, IL

DISTRIBUTION LIST (continued)

HQDA (DAMA-CSS)
Washington, DC 20310

PM, ARTADS
ATTN: DRCPM-TDS-TF
Fort Monmouth, NJ 07703

Commander
US Army TARADCOM
ATTN: DRSTA-RGD
ATTN: DRSTA-MST
Warren, MI 48090

Commander
US Army Aviation Systems Command
P.O. Box 209
ATTN: DRSAV-FPP
St. Louis, MO 63120

Commander
US Army Electronics Command
ATTN: DRSEL-SA
ATTN: DRSEL-TL-M
ATTN: DRSEL-TL-MS (14 cys)
ATTN: DRSEL-MA-D
ATTN: DRSEL-NL-BG
ATTN: DRSEL-MS-TI (2 cys)
Fort Monmouth, NJ 07703

Commander
Tobyhanna Army Depot
ATTN: DRXTO-MI-O
ATTN: Dep. Dir, Mr. W. Morris
Tobyhanna, PA 18466

Commander
US Army Materiel Systems
Analysis Agency
ATTN: DRXSY-CC
Aberdeen Proving Ground, MD 21005

Commander
MIRCOM
ATTN: DRSMI-RLE
ATTN: DRSMI-RLD
ATTN: Dir of Maint, TMDE Ofc
Redstone Arsenal
Huntsville, AL 35809

Commander
US Army Satellite Communications Agency
ATTN: DRCPM-SC-8B
Fort Monmouth, NJ 07703

Commander
US Army Maintenance Management Center
ATTN: DRXMD-TR
Lexington, KY 40507

Commander
US Army ARADMAC
ATTN: SAVAE-EMP
Corpus Christi, TX 78419

Commander
US Army Signals Warfare Lab
ATTN: DELSW-OS
Arlington Hall Station
Arlington, VA 22212

Commander
Sacramento Army Depot
ATTN: DRSXA-MPE
ATTN: Dep. Dir, Maint (Mr. A. Weaver)
Sacramento, CA 95813

Commander
US Army Logistics Center
ATTN: ATCL-MC
Fort Lee, VA 23801

Commander
Picatinny Arsenal
ATTN: TSD (Bldg 352)
Dover, NJ 07801

Commander
US Army Materiel Development and
Readiness Command
ATTN: DRCDL
ATTN: DRCDE-T
ATTN: DRCDE-DS
5001 Eisenhower Avenue
Alexandria, VA 22333

DISTRIBUTION LIST (continued)

Commander
Kelly Air Force Base
ATTN: SA-ALC/MMIN
ATTN: SA-ALC/XRXM
ATTN: SA-ALC/MAGT
San Antonio, TX 78241

Headquarters
AFSC/LGM
Andrews Air Force Base
ATTN: Mr. Clark Walker
Washington, DC 20334

Headquarters
USAF/LGYE
ATTN: Mr. C. Houk
Washington, DC 20330

Commander
Wright Patterson Air Force Base
ATTN: ASD/ENEGE
ATTN: ASD/AEG
Dayton, OH 45433

Headquarters
Naval Materiel Command
ATTN: Dir, Automatic Test
Equipment Management &
Technology Ofc (MAT 036T)
Washington, DC 20360

Commander
Naval Air Systems Command
ATTN: AIR 53424, Mr. M. Myles
Washington, DC 20360

Commander
Naval Electronic Systems
Engineering Center
ATTN: 03T, Mr. F. Fernandez
P.O. Box 89337
San Diego, CA 92138

Commander
Naval Electronic Systems Command
ATTN: ELEX 4802, Mr. G. Margulies
Washington, DC 20360

Commander
Naval Sea Systems Command
ATTN: SEA 9822B, Mr. John Yaroma
Washington, DC 20360

Commander
Naval Electronics Laboratory Center
ATTN: Code 4050, Mr. D. Douglas
271 Cataline Boulevard
San Diego, CA 92152

NASA
J. F. Kennedy Space Center
ATTN: DE-DEO-2, Mr. L. Dickison
Kennedy Space Center, FL 32899

RLG Associates, Inc.
11250 Roger Bacon Drive
Suite 16
Reston, VA 22090

Mr. Ralph Shirak
RCA Corporation
P.O. Box 588
Burlington, MA 01803

Mr. B. Richard Climie
Aeronautical Radio, Inc.
2551 Riva Road
Annapolis, MD 21401

Mr. Roger W. Fulling
Hughes Aircraft Company
1515 Wilson Boulevard
Arlington, VA 22209

Mr. Leon G. Stucki
McDonnell Douglas Astronautics Co.
5301 Bolsa Avenue
Huntington Beach, CA 92647

Dr. Noah S. Prywes
Dept. of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19174

DISTRIBUTION LIST (continued)

National Bureau of Standards
Bldg 225, RM A-331
ATTN: Mr. Leedy
Washington, DC 20231

Massachusetts Computer Associates
26 Princess Street
ATTN: Mr. D. Loveman
ATTN: Mr. Thomas Cheatham, Jr.
Wakefield, MA 01880

Mr. Robert G. Kurkjian
Hughes Aircraft Company
Bldg 6, MS E111
Culver City, CA 90230

Mr. Andrew Mills
Hewlett-Packard
974 East Arques
Sunnyvale, CA 94806

Mr. M. T. Schloesser
Lockheed-California Company
Div. of Lockheed Aircraft Corp.
Bldg 167
Burbank, CA 91520

Analytics
2500 Maryland Road
ATTN: Mr. S. Leibholtz
ATTN: Mr. R. Brachman
Willow Grove, PA 19090